# @CHIP-RTOS - General introduction

IPC@CHIP® Documentation **Index**

## Short general explanation




The name @CHIP-RTOS is used as the term for the operating system of the IPC@CHIP® (older manuals are still using the term BIOS). For an overview of the provided features and the general architecture of the operating system see **@CHIP-RTOS architecture**.

The operating system is able to execute multiple (up to 12) DOS programs simultaneous. Each program runs as a task of the RTOS kernel. These 16 bit DOS programs can be created with a variety of development tools. However, the instruction opcodes in these programs must be confined to the 80186 instruction set. For C/C++ programming, Beck offers the *Paradigm Beck IPC Edition* and the *Borland 5.02* development environments (see **www.beck-ipc.com**).

This document describes the Application Programming Interface (API) of the @CHIP-RTOS. It should be used as a reference manual for IPC@CHIP® application programmers. For a better understanding of this manual the reader should have some experience in programming DOS applications.

The document explains how to call every provided function of the @CHIP-RTOS from user application

programs. The API of the @CHIP-RTOS is structured into different sections (e.g. HARDWARE API, RTOS API, TCPIP API, ....). All sections are listed as HTML-links on the **main index page** of this document.

For every API section the @CHIP-RTOS provides a software interrupt with different function numbers for each API call. The software interrupt mechanism is the standard way for DOS programs to call internal functions of the operating system (e.g. file operations, reading/printing characters, etc.).

A C-Library is available which provides an efficient interface to the set of underlying @CHIP-RTOS services documented here. Usage of this Beck C-Library is highly recommended. Both binary and source for the Beck C-library are provided at our download page, **www.beck-ipc.com**. A collection of example programs is also provided, which demonstrate the usage of the API.

Before calling the specified software interrupt, every necessary function parameter must be loaded into the processor registers. Each API call description details which parameters are required in the registers. The @CHIP-RTOS uses the parameters from the processor registers and executes the corresponding function. After the software interrupt execution, some specified processor registers contain the return result(s) from the function. The meaning of the return result(s) are specified with each API call description.

You will find some useful general notes about programming DOS applications for the IPC@CHIP® under **Programming notes**.

For some necessary knowledge about hardware details of the IPC@CHIP®, see the hardware manuals available from the download area of **www.beck-ipc.com**.

---

End of document

---




# IPC@CHIP® RTOS Documentation [Build 29.04.2008]

*RTOS Documentation Index:*

| | | |
|---|---|---|
| Introduction | Multitasking introduction | Overview |
| Scaled variants | Performance comparison | Startup initialization |
| Security notes | Programming notes | Boot Flow Chart |

*Command Shell and System configuration:*

Command Shell: Description of the command processor
AUTOEXEC.BAT: Automatic execution of applications on system start
CHIP.INI: System configuration based on CHIP.INI file

*CLIB Documentation Index:*

API Library Documentation (C-Libs)

*Software Interrupt Documentation Index:*

API Documentation (Interrupts)

*Server descriptions:*

PPP Interface: How to configure the IPC@CHIP® PPP server
Web server: Short description of the IPC@CHIP® Web server
FTP server: Short description of the IPC@CHIP® FTP server
TFTP server: Short description of the IPC@CHIP® TFTP server
Telnet server: Short description of the IPC@CHIP® Telnet server
SSH Server: Secure Shell connections

---

## Applies to all @CHIP-RTOS versions lower than or equal to:

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.22 | V1.22 |

---

End of document

---




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS - Multitasking with @Chip-RTOS

IPC@CHIP® Documentation **Index**

Some common explanations about multitasking with the @CHIP-RTOS

**MULTITASKING INTRODUCTION**

- **Abbreviations Used**
- **REASONS FOR USING MULTITASKING**
  - **Different Priority Work**
  - **Event Triggered Actions**
- **REASONS NOT TO USE MULTITASKING**
  - **Resources Expended per Task**
  - **Critical Sections**

**PRIMARY TASK ATTRIBUTES**

- **TASK PRIORITY**
- **TASK STATE**

**DOS PROGRAM TASKS**

**SYSTEM TIMING**

- **TIME-SLICING**
- **PERIODIC TASKS**
  - **Roughly Periodic**
  - **Precisely Periodic**

**CRITICAL SECTIONS IN PROGRAMS**

- **EXAMPLE CRITICAL SECTION**

**PROTECTING CRITICAL SECTIONS**

**Semaphore Protection**

**Interrupt Masking**

**RTOS Task Switch Lock**

**Critical Section Protection Methods Summary**

**CONTROL AND COMMUNICATION BETWEEN TASKS**

**IPC@CHIP® Documentation Index**

---

# Multitasking Introduction

The @Chip-RTOS provides for a multitasking operation. A task provides a thread of execution. Each task has its own context, including an instruction pointer and program stack. Each task in the @Chip-RTOS has a unique priority of execution, providing a preemptive form of multitasking. By executing multiple tasks at the same time, various activities can be performed concurrently.

## Abbreviations Used

| Acronym | Stands For |
|---|---|
| API | Application Programming Interface |
| ISR | Interrupt Service Routine |
| RTI | Real-Time Interrupt |
| RTOS | Real-Time Operating System |
| TCP/IP | Transport Control Protocol / Internet Protocol |

**Table 1 ) Abbreviations**

## Reasons for using Multitasking

Here are some situations where multitasking can be helpful.

### Different Priority Work

Probably the most compelling reason for using multitasking is when required activities have different priorities. For example, an application with a user interface may need to be responsive to keyboard entry from a user console while at the same time the program is conducting a time consuming calculation or data base search. A low priority task could perform the calculation/data base search while a high priority task periodically polls (or sleeps waiting) for keyboard input. Typically there would

be some communication between these two tasks, such as having the keyboard task cancel the background data base search in the event that the user presses the escape key.

In more involved applications, there might be a complete spectrum of concurrent activity at different priorities. Just for an example, the set of tasks within a real-time application might be as follows. Listed in priority order:

- 50 Hz Data Acquisition Loop (top priority)
- 10 Hz Data Pre-processing Loop
- 1 Hz Data Processing Loop
- 0.1 Hz Kalman Filter
- Background Loop for Integrity Self-test

### Event Triggered Actions

Sometimes an activity is required only upon an event such as some data becoming available. For these situations a task could be made to block, awaiting the event or data arrival. For example, a task could make a blocking call on a TCP/IP port with the recv() library function. The task sleeps until some data arrives at the respective socket, or until a prescribed time-out period expires.

Note that polling can usually be an alternative software design approach, as opposed to applying a dedicated task to each event or data source.

## Reasons not to use Multitasking

While multitasking can be an ideal solution for some applications, it also has its disadvantages. Some of these disadvantages are noted here. The point is that you should not complicate an application with multiple tasks unless to do so truly makes your application simpler.

### Resources Expended per Task

Each task requires its own stack space. This stack must be made large enough to support the system's interrupt handlers, some of which operate without a switch to a system stack. A minimum stack space of 1024 bytes is recommended.

### Critical Sections

Objects (memory or devices) shared between tasks can lead to critical sections. A critical section is a section of code in a task which is sensitive to order of execution relative to code in some other task, such that there is a possible (prior to adding the necessary protection) firing order of the two instruction streams which leads to incorrect results.

When you design your application with only a single task, it is safe to say you have no critical sections. When more than one task is used, **you must beware**. Due to the importance of understanding and recognizing critical sections in multitasking systems, this topic is covered in more detail below on chapter Critical Sections in Programs.

# Primary Task Attributes

Two important attributes of each task are task priority and state.

## Task Priority

Each task executing under the @Chip-RTOS has a unique internal task priority. This internal task priority is represented with a 16 bit value, the most significant byte of which is the user task priority which is set at task creation time or using the RTX_Change_Task_Prio() API (and is visible through the RTX_Get_Task_State() API). The hidden least significant byte of the task priority is used internally by the @Chip-RTOS to assign each task at a given user priority a unique priority by appending a sequence number to the upper byte.

A task is assigned the lowest internal priority of all tasks with the same user priority whenever that task is appended to the list of tasks at that given user priority. This occurs when:

a) the task is created

b) the task's priority is changed

c) the task's time-slice period times out (see chapter Time-Slicing).

Application program tasks can range in user priority from 3 to 127 (inclusive), where 3 is higher priority. Generally, user task priorities between 20 and 30 are recommended. This recommendation is based on the priority assignments of the built-in system tasks. Too high a priority for an application task may block urgent system tasks: e.g. the Ethernet receiver task.

## Task State

In Table 2 below, the possible states and sub-states for @Chip-RTOS tasks are summarized. There are three primary states: Active, Blocked and Suspended.

| State | Sub-State | Notes and State Transitions |
|---|---|---|
| **Active** | Executing | Highest priority non-waiting task |
| | Pending | In queue ordered by task priority |
| **Blocked** | Trigger Wait[1] | RTX_Restart_Task() → **Active** |
| | Semaphore Wait[2] | RTX_Signal_Sem(), RTX_Release_Sem() → **Active** |
| | Event Group Wait[2] | RTX_Signal_Events() → **Active** |
| | Message Exchange Wait[2] | RTX_Send_Msg() → **Active** |
| | Asleep[2] | RTX_Wakeup() → **Active** |
| **Suspended** | Free to run | RTX_Resume_Task() → **Active** |

| | | |
|---|---|---|
| | Trigger Wait[1] | RTX_Resume_Task() → **Blocked**<br>RTX_Restart_Task() → , Free to run[3] |
| | Semaphore Wait[2] | RTX_Resume_Task() → **Blocked**<br>Granted semaphore → , Free to run[3] |
| | Event Group Wait[2] | RTX_Resume_Task() → **Blocked**<br>RTX_Signal_Events() → , Free to run[3] |
| | Message Exchange Wait[2] | RTX_Resume_Task() → **Blocked**<br>RTX_Send_Msg() → , Free to run[3] |
| | Asleep2 | RTX_Resume_Task() → **Blocked**<br>RTX_Wakeup() → , Free to run[3] |

1) - Trigger Wait sub-state is entered after RTX_Create_Task_Without_Start() or after a task has terminated.
2) - A specified time-out period in milliseconds can be applied to these states.
3) - Only the sub-state has changed here.

**Table 2 ) @Chip-RTOS Task States**

The set of active tasks we speak of as executing concurrently. However, only a single task (at most[1]) is executing at any given time since the IPC@CHIP® contains only a single CPU. The task selected for execution by the @Chip-RTOS will always be the highest priority of the tasks that are in the active state.

The C-library routines which force a task to exit the Blocked and Suspended states when called by some other task or Interrupt Service Routine (ISR) are stated in the table. The two inactive states, Blocked and Suspended, differ in their exit state transitions. The RTX_Suspend_Task() API transitions a task into the Suspended state.

1 Hardware interrupt service routines can momentarily suspend the executing task.

---

**Top of this document**
**IPC@CHIP® Documentation Index**

# DOS Program Tasks

Each DOS program is launched as a task under @Chip-RTOS. These tasks are created with initial priority 25 and time-slicing disabled. Within these DOS programs, users can create additional tasks with the RTX_Task_Create() or RTX_Task_Create_Without_Run() API

---

# System Timing

The @Chip-RTOS uses a 1000 Hz Real-Time Interrupt (RTI) for its time base. Therefore one millisecond is the lower resolution available for task timing.

Users can install Timer Callback procedures with the RTX_Install_Timer() API. Your callback procedure is invoked within the top priority kernel task at a specified interval.

# Time-Slicing

For the tasks created within DOS programs by the user, a time-slicing feature is available. This feature is enabled for a specific task by specifying a non-zero number of milliseconds in the time_slice member of the TaskDefBlock structure passed to the RTX_Task_Create() or RTX_Task_Create_Without_Run() functions.

A time-sliced task will be permitted to execute for time_slice milliseconds (RTI ticks) after which time it will be cycled to the end of the list of tasks at this task's user priority. (The task is not charged for ticks during which it was blocked, suspended or active-pending preempted by some higher priority task.) In the special case where it is the only active task at that user priority, it would then on time-out immediately be given another time_slice milliseconds execution time budget and allowed to continue execution. Otherwise one of the other active tasks pending execution at this same user priority will begin execution and the previously executing task whose time-slice expired will be cycled to the end of the priority queue in a round-robin fashion. Note that the next task to execute may or may not be configured for time_slice operation.

The time-slice operation would apply primarily to fully independent tasks which do not pass any data between each other. Time-slicing can introduce chaos into a program which could execute more orderly using explicit yields (e.g. RTX_Sleep_Time API). The extra task switching due to time-slice operation can cause critical sections to appear where they otherwise would not if the task was permitted to execute up to where the program yields voluntarily.

There may be times where time-slicing is the graceful design solution, but reliance on this technique raises the suspicion that the software design was not thought out thoroughly. Also keep in mind that any task with lower user priority than the time-sliced tasks will never be executed so long as any of the time-sliced tasks are active. During execution of a time-sliced task, there is a very slight additional load placed on the system's 1000 Hz RTI.

# Periodic Tasks

Periodic tasks can be created in either of two ways, depending on how accurate the execution period is required to be for the respective application.

## Roughly Periodic

The simplest form for a periodic task uses a RTX_Sleep_Time call within a loop as shown below in Figure 1. The period of this loop will not be exact, but would be close enough for many applications.

```
#define SLEEP_10HZ  (90)  // Assuming 10 ms CPU load per 100 ms

void huge Task_Roughly_10Hz(void)
{
    while (1)
    {
         Activity_10Hz() ;   // Get here about each 100 ms
         RTX_Sleep_Time(SLEEP_10HZ) ;
    }
}
```

**Figure 1 ) Sleep Based Periodic Loop**

The SLEEP_10HZ constant used in this example is adjusted based on the expected system loading,

including CPU dwell within the Activity_10Hz procedure. This would require some timing measurements to be made during the program's development.

## Precisely Periodic

A precisely periodic loop can be controlled with an RTOS timer. This will result in a periodic loop which on average tracks the CPU quartz clock. A RTOS timer periodically wakes up the periodic task loop as illustrated below in Figure 2.

```
static int TaskID_10Hz ;

void huge Task_10Hz(void)
{
    while (1)           // 10 Hz loop
    {
         RTX_Sleep_Request () ;
         Activity_10Hz() ;   // Get here each 100 ms.
    }
}

static void huge Timer_Callback(void) // Clean 10 Hz
{
    RTX_Wakeup (TaskID_10Hz) ;
}

static TimerID ;
static TimerProc_Structure Timer_Spec = {
    &TimerID,
    Timer_Callback,
    0,
    { '1', '0', 'H', 'z'},
    100              // .interval = 100 milliseconds
} ;

void Initialize(void)
{
    extern TaskDefBlock Task_10Hz_Def ;
    RTX_Create_Task (&TaskID_10Hz, &Task_10Hz_Def) ;
    RTX_Install_Timer (&Timer_Spec) ;
    RTX_Start_Timer (TimerID) ;
}
```

**Figure 2 ) Timer Based Periodic Loop**

An alternative method here would be to use RTX_Suspend_Task in Task_10Hz and RTX_Resume_Task in Timer_Callback. However, the RTX_Wakeup has the advantage of a wakeup pending flag used in the implementation which covers for the case where, due to CPU loading, the Task_10Hz may not yet have reached the RTX_Sleep_Request call before the RTX_Wakeup is executed in the Timer_Callback. In this case when the RTX_Sleep_Request is later called in Task_10Hz, the API will return immediately after clearing the task's internal wakeup pending flag, which had been set when Timer_Callback called RTX_Wakeup before Task_10Hz reached its sleep.

---

**Top of this document**
**IPC@CHIP® Documentation Index**

# Critical Sections in Programs

When multitasking is used, the programmer must beware of critical sections which may occur between threads.

Critical sections can be protected with proper design. The important step at program design time is to identify these code sections, which are not always obvious. Most programmers are well aware of what critical sections are. However, due to their importance when multitasking, a simple example is provided here for emphasis.

**Paradigm-Beck IPC Edition compiler users note:** The program heap manager behind the `malloc` standard library function contains critical sections. Beck offers a multithreaded version of **malloc** which is compatible with the Paradigm-Beck IPC Edition compiler's run-time library.

## Example Critical Section

Data sharing between tasks leads to critical sections when the shared data object can not be read or written atomically, within a single non-interruptible machine instruction.

```
static unsigned long My_Ticker = 1 ;

void huge Task_A(void)
{
    if (My_Ticker < 0x7FFFFFFFL)
    {
       My_Ticker++ ;
        //  Borland compiler's machine code:
        //     ADD    My_Ticker,1H   ; Increment LS 16 bits
        // → Sensitive to task switch here!
        //     ADC    My_Ticker+2,0H ; Carry into MS 16 bits
    }
}


void huge Task_B(void)
{
    if (My_Ticker == 0)
    {
        Surprised_to_be_Here() ;  // How did this happen?
    }
}
```

**Figure 3 ) Critical Section Example**

After a brief review of the C code in the above example, the C programmer might suspect a hardware problem if the Surprised_to_be_Here() function was to ever execute. However, with a closer examination of the resulting machine assembly code and multitasking consideration, we will see that execution of the Surprised_to_be_Here() function is possible[2].

2 And "if something bad can happen, it will happen".

All tasks in the @Chip-RTOS system have a unique task priority. So in the above example either Task_A can potentially interrupt execution of Task_B, or visa-versa, depending on the assigned priorities. Consider the case where priority of Task_A is lower (higher numerically) than priority of Task_B, such that Task_B

can preempt Task_A. This case can lead to execution of the Surprised_to_be_Here() function under the following circumstances.

Let us say that Task_A has already executed 0xFFFE times and on its 0xFFFF'th execution it is preempted by Task_B at the indicated "Sensitive" point immediately after executing the ADD opcode which increments the lower half of the 32 bit up counter. At this exact point, the My_Ticker value will read zero due to the carry from the lower 16 bits not yet being applied to the upper half word. And thus Task_B lands in the Surprised_to_be_Here() function when it encounters the half updated My_Ticker reading.

## Protecting Critical Sections

Three methods for protecting critical sections are presented here.

1) Semaphore

2) Interrupt Masking

3) RTOS Task Switch Lock

Each method has its advantages and limitations, which are summarized at the end of this discussion. The choice of which method to use will depend on the design situation.

### Semaphore Protection

A common way to protect critical sections is with the use of semaphores. The @Chip-RTOS provides resource semaphores which provide a mutually exclusive access to a resource or data object.

The example defective code from Figure 3 above can be corrected with the use of a resource semaphore as shown below.

```
static unsigned long My_Ticker = 1 ;

void huge Task_A(void)
{
    RTX_Reserve_Sem(semID, 0) ;
    if (My_Ticker < 0x7FFFFFFFL)
    {
       My_Ticker++ ;
    }
    RTX_Release_Sem(semID) ;
}


void huge Task_B(void)
{
    unsigned long ticker ;
    RTX_Reserve_Sem(semID, 0) ;
    ticker = My_Ticker ;
    RTX_Release_Sem(semID) ;
    if (ticker == 0)
    {
          Surprised_to_be_Here() ;  // How did this happen?
    }
```

```
}
```

**Figure 4 ) Protected Critical Section with Semaphore**

Now the Surprised_to_be_Here() function will never be executed.

A potential disadvantage to using semaphores is a possible task priority inversion, where a high priority task is blocked by a lower priority task as it awaits the semaphore. To illustrate this point, consider an example where task priorities are designed as follows:

Task_A - Priority 60 (low priority)

Task_B - Priority 4 (very high priority)

If Task_A is suspended while it has possession of the semaphore, Task_B will have to wait if it then tries to access the same semaphore at that moment. This wait is effectively at the very low priority 60, which would mean that Task_B (priority 4) must sit waiting behind time consuming system activities such as FTP transfer (priority 41). In applications where this potential priority inversion is not acceptable, either the interrupt masking or task lock methods of protecting critical sections discussed below can be considered as an alternative to using semaphores.

## Interrupt Masking

Interrupt masking can in some cases be a safe alternative to using semaphores to protect critical sections. This fast method places a minimum load on the system, so is most suitable where performance is a concern. The interrupt masking method is used in the example below.

```
define MASK_INTERRUPTS   asm{CLI}
#define ENABLE_INTERRUPTS asm{STI}

static unsigned long My_Ticker = 1 ;

void huge Task_A(void)
{
    MASK_INTERRUPTS ;   // Needed if Task_A is lower priority
    if (My_Ticker < 0x7FFFFFFFL)
    {
       My_Ticker++ ;
    }
    ENABLE_INTERRUPTS ;
}


void huge Task_B(void)
{
    unsigned long ticker ;
    MASK_INTERRUPTS ;  // Needed if Task_B is lower priority
    ticker = My_Ticker ;
    ENABLE_INTERRUPTS ;
    if (ticker == 0)
    {
          Surprised_to_be_Here() ;  // How did this happen?
    }
}
```

**Figure 5 ) Protected Critical Section with Interrupt Masking**

This method of protection is safe to use when the section being protected executes in very few machine cycles, as is the case in this example. The concern is the hardware interrupt latency created by this interrupt mask period. Masking interrupts for as long as 50 microseconds should be tolerable on most systems. Caution must be used to assure that interrupts are always quickly re-enabled when ever they are disabled!

Note that when the nature of the two tasks competing for access to the resource (Task_A and Task_B in this example) dictates that one is higher priority than the other, only the lower priority task requires the interrupt masking. It is not possible that the lower priority task could preempt the higher priority task (unless the program design was to change task priorities dynamically somewhere).

## RTOS Task Switch Lock

A further alternative to using semaphores to protect critical sections is to prevent task switching within the critical section. This method is shown in the example below.

```
static unsigned long My_Ticker = 1 ;

void huge Task_A(void)
{
      // Disable needed if Task_A is lower priority
    RTX_Disable_Task_Scheduling() ;  // Task switch lock
    if (My_Ticker < 0x7FFFFFFFL)
    {
       My_Ticker++ ;
    }
    RTX_Enable_Task_Scheduling() ;   // Resume task switching
}


void huge Task_B(void)
{
    unsigned long ticker ;
      // Disable needed if Task_B is lower priority
    RTX_Disable_Task_Scheduling() ;  // Task switch lock
    ticker = My_Ticker ;
    RTX_Enable_Task_Scheduling() ;   // Resume task switching
    if (ticker == 0)
    {
          Surprised_to_be_Here() ;  // How did this happen?
    }
}
```

**Figure 6 ) Protected Critical Section with Task Lock**

Hardware interrupts continue to be serviced during the task lock, so you can include more work now within the critical section than was possible with the interrupt masking method. However, the task lock period should still be keep to some reasonable small amount of time. Note that task locks also inhibit all system timer activity.

### Critical Section Protection Methods Summary

The design trade-offs for the three methods presented above for protecting critical sections are summarized in Table 3.

| Method | Advantage | Limitations |
|---|---|---|
| Semaphore | A long duration of critical section does not adversely affect portions of system not accessing the semaphore. | Can result in a priority inversion. |
| Interrupt Mask | Most efficient of all three methods. (Executes quickly!)<br>No priority inversion | Impact on system operation becomes a concern if interrupt mask time can exceed around 50 us. |
| Task Lock | No priority inversion. | Impact on system operation becomes questionable if task lock duration exceeds around 400 us. |

**Table 3 ) Critical Section Protection Methods**

---

**Top of this document**
**IPC@CHIP® Documentation Index**

# Control and Communication between Tasks

The @Chip-RTOS provides the following mechanisms for tasks to control one another and to communicate. These interactions can either be between tasks within the same DOS application, or across applications.

- Semaphores
- Event Groups
- Message Exchanges

The usage of these @Chip-RTOS resources is covered in depth within the on-line HTML help.

---

**Top of this document**
**IPC@CHIP® Documentation Index**

---

End of document

---

# @CHIP-RTOS - Software Overview



## @CHIP-RTOS features

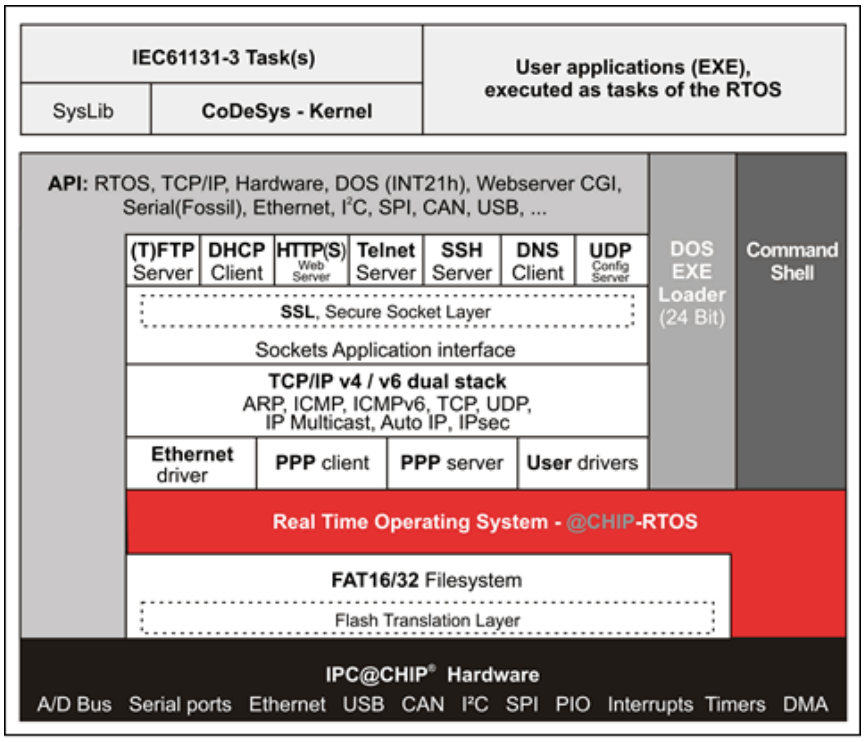
| RTOS | SC1x | SC1x3 SC23 |
|---|---|---|
| Tasks | 35 | 78 |
| Sum of Semaphores, Timer, Event goups | 78 | 128 |
| Message exchanges | 10 | 32 |

These are the entire RTOS resources. Some of them are used by the RTOS itself. See RTOS API documentation for available user resources.

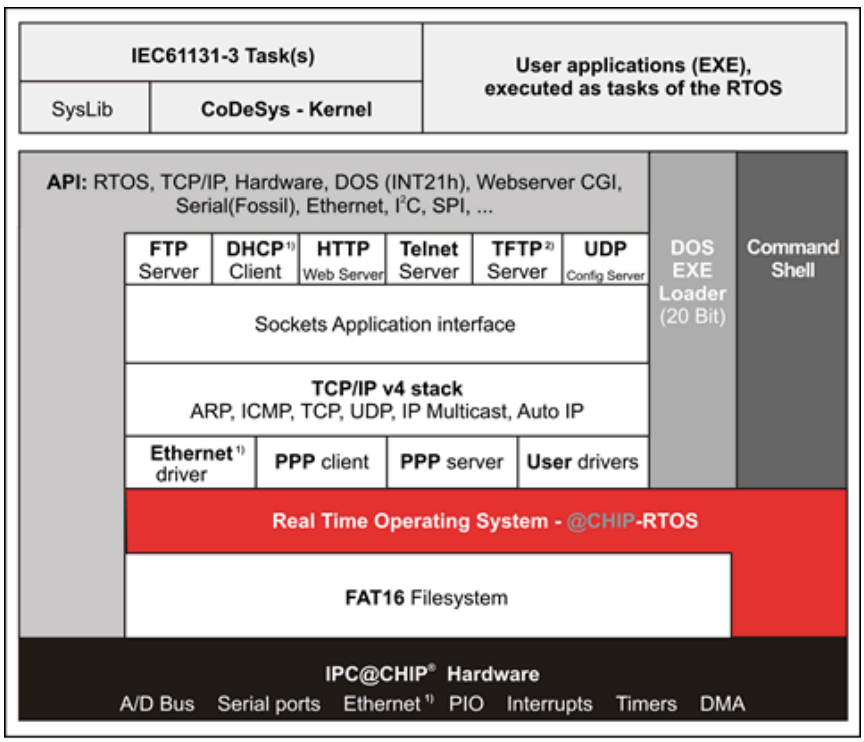
| RTOS filesystem | SC1x | SC1x3 SC23 |
|---|---|---|
| Internal RAM disk | x | x |
| Internal Flash disk | x | x |
| Flash translation layer | - | x |
| External disk drives | 1 | 2 |
| FAT16 | x | x |
| FAT32 | - | x |
| Long file names | - | x |
| Tasks with file access | 15 | 15 |
| Maximum open files | 20 | 20 |

| TCP/IP | SC1x | SC1x3 SC23 |
|---|---|---|
| IPv4 Stack | x | x |
| IPv6 Stack | - | x |
| TCP | x | x |
| UDP | x | x |
| ARP | x | x |
| ICMP | x | x |
| IP Multicast | x | x |
| Auto IP | x | x |
| Socket interface | x | x |
| SSL | - | x |
| IPsec/IKE | - | x |
| Sockets | 64 | 256 |
| Internal devices interfaces | | |
| - Ethernet | x | x |
| - PPP server | x | x |
| - PPP client | x | x |
| - Loopback | x | x |

| TCP/IP applications | SC1x | SC1x3 SC23 |
|---|---|---|
| HTTP webserver | x | x |
| SSL webserver | - | x |
| FTP server | x | x |
| Telnet server | x | x |
| SSH server | - | x |
| DHCP client | x | x |
| DNS client | - | x |
| TFTP server | x | x |
| SNMP MIB variable support | x | x |
| UDP config server for @CHIP-RTOS upgrade | x | x |




@CHIP-RTOS architecture (SC1x3)




1) This feature is not available on SC11

2) This feature is not part of the standard SC1x RTOS versions

@CHIP-RTOS architecture (SC1x)

**DOS-EXE Loader**

Up to 12 DOS application programs can run as tasks of the RTOS. Multiple tasks can be created inside one DOS-EXE.

**DOS-like command shell**

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

**CoDeSys IEC61131-3 Kernel**

Runtime license for CoDeSys IEC61131-3 available

**Upgrade**

@CHIP-RTOS Upgrade via Ethernet/UDP or serial Bootstrap loader

**XMODEM**

Serial file transfer via xmodem protocol

| Application Programming Interface | SC1x | SC1x3 SC23 |
|---|---|---|
| RTOS services | x | x |
| TCP/IP socket interface | x | x |
| DOS interrupt 21h and others | x | x |
| Webserver CGI | x | x |
| Hardware | x | x |
| I2C | sw | hw |
| SPI | sw | hw |
| CAN | - | x |
| USB host/device | - | x |
| Serial devices (Fossil interface) | x | x |
| Ethernet packet driver (for NON-TCP/IP purposes) | x | x |
| Special @CHIP-RTOS services | x | x |

| Application examples (DOS EXE) | SC1x | SC1x3 SC23 |
|---|---|---|
| RTOS API examples | x | x |
| TCP/IP API examples | | |
| - FTP client | x | x |
| - HTTP client | x | x |
| - Other TCP/IP examples | x | x |
| Webserver CGI examples | x | x |
| External IDE disk drivers | x | x |
| Hardware API examples | x | x |
| I2C examples | x | x |
| SPI examples | x | x |
| CAN examples | x | x |
| USB examples | - | x |
| Fossil examples | x | x |
| ModBus examples (TCP and serial) | x | x |
| PPPoE driver | - | x |
| API C-Libraries | x | x |

End of document




# @CHIP-RTOS - Scalable @CHIP-RTOS variants for the IPC@CHIP®



The @CHIP-RTOS for the IPC@CHIP®s SC11/SC12/SC13 is provided by 6 official variants with a different set of included features. The variants with their provided features and the available RAM and Flash memory sizes are listed at the tables below.

- SC11Vxxxx_TINY.hex
- SC11Vxxxx_SMALL.hex
- SC11Vxxxx_MEDIUM.hex
- SC11Vxxxx_LARGE.hex
- SC11Vxxxx_MEDIUM_PPP.hex
- SC11Vxxxx_LARGE_PPP.hex
- SC12Vxxxx_TINY.hex
- SC12Vxxxx_SMALL.hex
- SC12Vxxxx_MEDIUM.hex
- SC12Vxxxx_LARGE.hex
- SC12Vxxxx_MEDIUM_PPP.hex
- SC12Vxxxx_LARGE_PPP.hex
- SC13Vxxxx_TINY.hex
- SC13Vxxxx_SMALL.hex
- SC13Vxxxx_MEDIUM.hex
- SC13Vxxxx_LARGE.hex
- SC13Vxxxx_MEDIUM_PPP.hex
- SC13Vxxxx_LARGE_PPP.hex

The @CHIP-RTOS for the IPC@CHIP®s SC1x3 and SC23 is provided by only one version, there are no variants. The SC1x3 and SC23 @CHIP-RTOS is called FULL and includes all available features.

- SC1x3Vxxxx_FULL.hex
- SC23Vxxxx_FULL.hex

| **Feature** | Tiny | Small | Medium | Medium PPP | Large | Large PPP | Full |
|---|---|---|---|---|---|---|---|
| RTOS kernel | x | x | x | x | x | x | x |
| Serial port fossil driver | x | x | x | x | x | x | x |
| RTOS filesystem | x | x | x | x | x | x | x |
| Ext Disk | | | x | x | x | x | x |
| XMODEM filetransfer | x | x | x | x | x | x | x |
| TCP/IP Ethernet driver | | x | x | x | x | x | x |
| TCP/IP PPP Client/server | | | | x | | x | x |
| Ethernet packet interface | x | x | x | x | x | x | x |
| TCP/IP protocol stack | | x | x | x | x | x | x |
| DHCP client | | x | x | x | x | x | x |
| I2C API | x | x | x | x | x | x | x |
| SPI API | x | x | x | x | x | x | x |
| Hardware API | x | x | x | x | x | x | x |
| CFG server | | x | x | x | x | x | x |
| Webserver | | | | | x | x | x |
| SSL Webserver | | | | | | | x |
| TFTP server | | | | | | | x |
| FTP server | | | x | x | x | x | x |
| Telnet server | | | x | x | x | x | x |
| SSH server | | | | | | | x |
| SNMP MIB support | | x | x | x | x | x | x |
| USB API | | | | | | | x |
| CAN API | | | | | | | x |

| | |
|---|---|
| IPsec API | x |

Here are the sizes of available RAM and flash disk for the different variants of the current @CHIP-RTOS version

| **SC11 V1.25** | Available RAM (KB) | Available Flash memory (KB) |
|---|---|---|
| Tiny | 467 | 403 |
| Small | 371 | 293 |
| Medium | 366 | 262 |
| Medium_PPP | 362 | 221 |
| Large | 340 | 238 |
| Large_PPP | 336 | 197 |

| **SC12 V1.25** | Available RAM (KB) | Available Flash memory (KB) |
|---|---|---|
| Tiny | 463 | 412 |
| Small | 366 | 299 |
| Medium | 362 | 268 |
| Medium_PPP | 358 | 226 |
| Large | 336 | 244 |
| Large_PPP | 332 | 202 |

| **SC13 V1.25** | Available RAM (KB) | Available Flash memory (KB) |
|---|---|---|
| Tiny | 464 | 398 |
| Small | 368 | 285 |
| Medium | 363 | 254 |
| Medium_PPP | 359 | 212 |
| Large | 337 | 230 |
| Large_PPP | 334 | 188 |

| **SC123 V1.20** | Available RAM (KB) | Available Flash memory (KB) |
|---|---|---|
| Full | 6987 | 946 |

| **SC143 V1.20** | Available RAM (KB) | Available Flash memory (KB) |
|---|---|---|
| Full | 6987 | 6832 |

| **SC23 V1.20** | Available RAM (KB) | Available Flash memory (KB) |
|---|---|---|
| Full | 6989 | 946 |

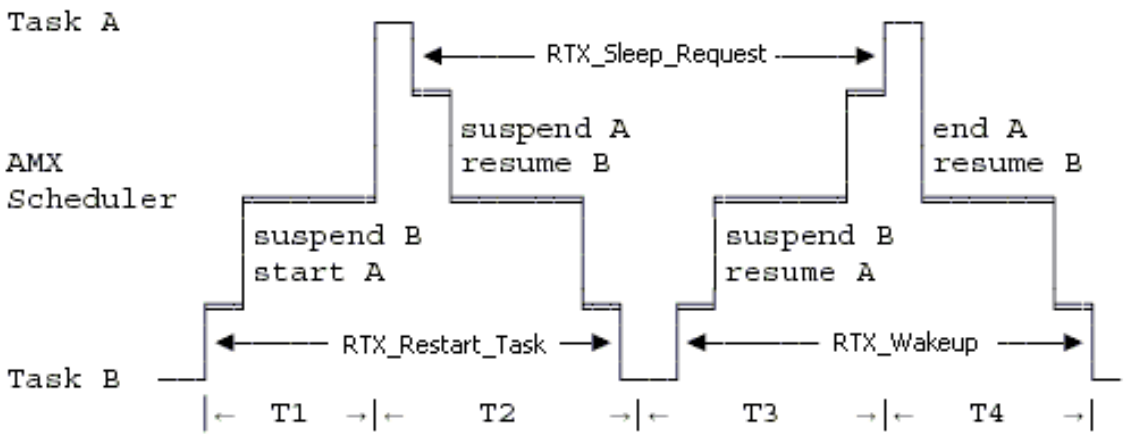
End of document

# @CHIP-RTOS - Performance comparision between IPC@CHIP® family



**RTOS task switch time**

The following diagramm and C code shows how the measured values were calculated.




```
void huge taskA(void)
{
   RTX_Sleep_Request();
}

void huge taskB(void)
{
   RTX_Restart_Task (taskAID);
   RTX_Wakeup (taskAID);
}

void main (void)
{
   //...
   result = RTX_Create_Task_Without_Run(&taskAID , &taskAdefblock);
   RTX_Sleep_Time(10);
   result = RTX_Create_Task(&taskBID , &taskBdefblock);
   //...
}
```

| | T1 | T2 | T3 | T4 |
|---|---|---|---|---|
| **SC12** | 104 µs | 80 µs | 100 µs | 60 µs |
| **SC13/SC11** | 35 µs | 35 µs | 36 µs | 26 µs |

| | | | | |
|---|---|---|---|---|
| **SC13-LF/SC11-LF** | 36 µs | 40 µs | 42 µs | 29 µs |
| **SC123/SC143/SC23** | 8,2 µs | 8,2 µs | 9,0 µs | 6,4 µs |

---

#### Interrupt latency

The interrupt handler latency is the typical time from the processor's first response to an interrupt request signal through to the first useful instruction inside of the user interrupt service procedure.

The user interrupt service procedure can be of type HW API or RTX.

| | **HW API handler latency** | **RTX handler latency** |
|---|---|---|
| **SC12** | 84 µs | 90 µs |
| **SC13/SC11** | 21 µs | 25 µs |
| **SC13-LF/SC11-LF** | 22 µs | 26 µs |
| **SC123/SC143/SC23** | 6,5 µs | 7,3 µs |

---

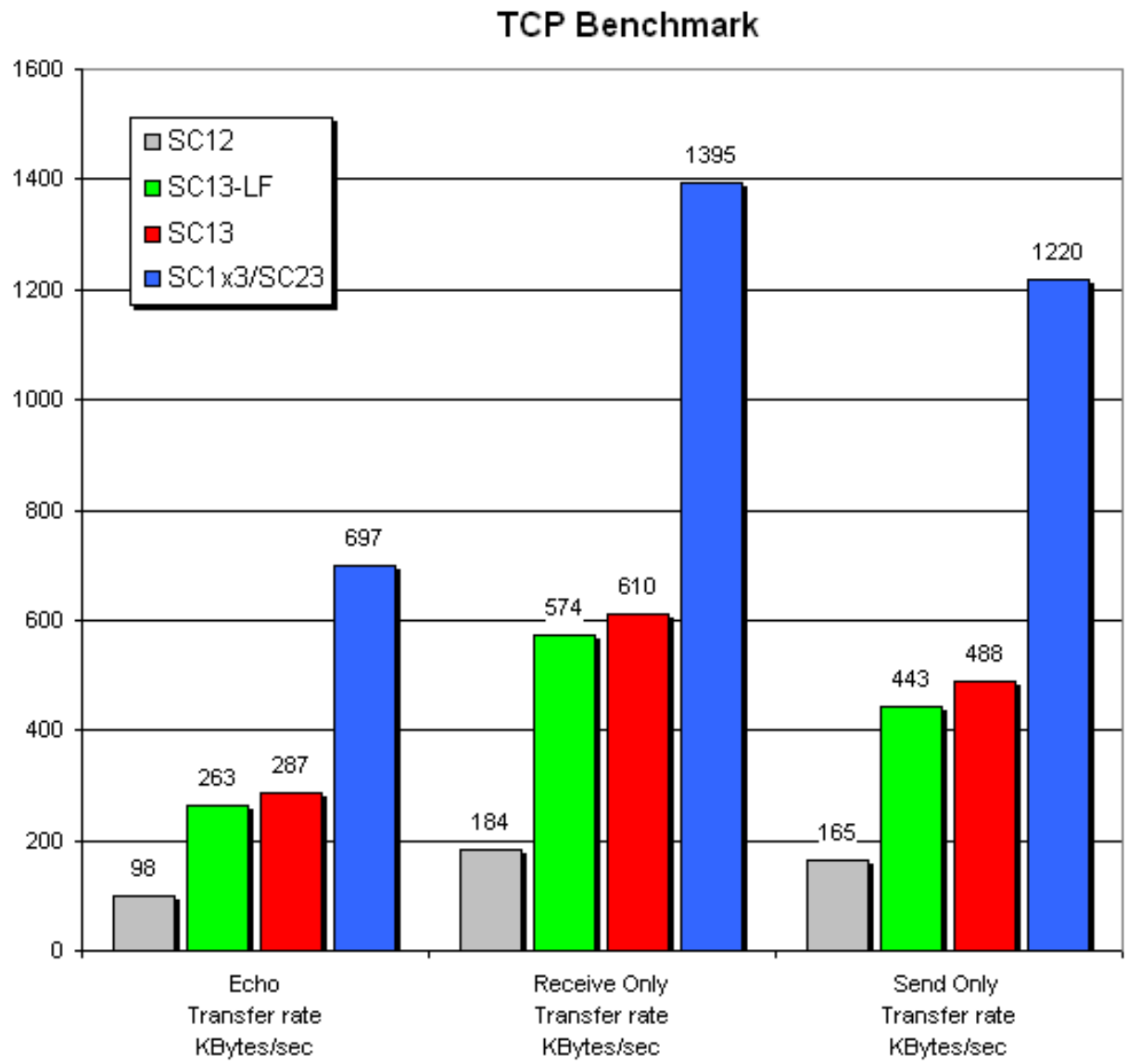
#### TCP echo

A TCP Echo client application, running on WinXP (Intel Pentium IV, 2,8 GHz) establishes a TCP connection to an IPC@CHIP® (SC12, SC13, SC13-LF, SC123/SC143, SC23) TCP Echo server application. The TCP server echoes the incoming data back to the client. The client application measures the response time and calculates the amount of databytes in KBytes/sec, which the IPC@CHIP® is able to echo back to the client.
The tests were made with a datasize of 10 MBytes.

Echotest : Measure the throughput of sending and receiving echo data.
Receive only: Measure the throughput only of receiving data from IPC@CHIP®.
Send only : Measures the throughput, if the IPC@CHIP® only sends data.

This test was made under special test conditions. We cannot guarantee that the measured performance results are achievable on other test environment and conditions.

| | **Echo transfer rate KByte/s** | **Receive only transfer rate KByte/s** | **Send only transfer rate KByte/s** |
|---|---|---|---|
| **SC12<br>10 Mbit/s, RTOS V1.25** | 98 | 184 | 165 |
| **SC13-LF<br>100 Mbit/s, RTOS V1.25** | 263 | 574 | 443 |
| **SC13<br>100 Mbit/s, RTOS V1.25** | 287 | 610 | 488 |
| **SC123/SC143/SC23<br>100 Mbit/s, RTOS V1.20** | 697 | 1395 | 1220 |




---

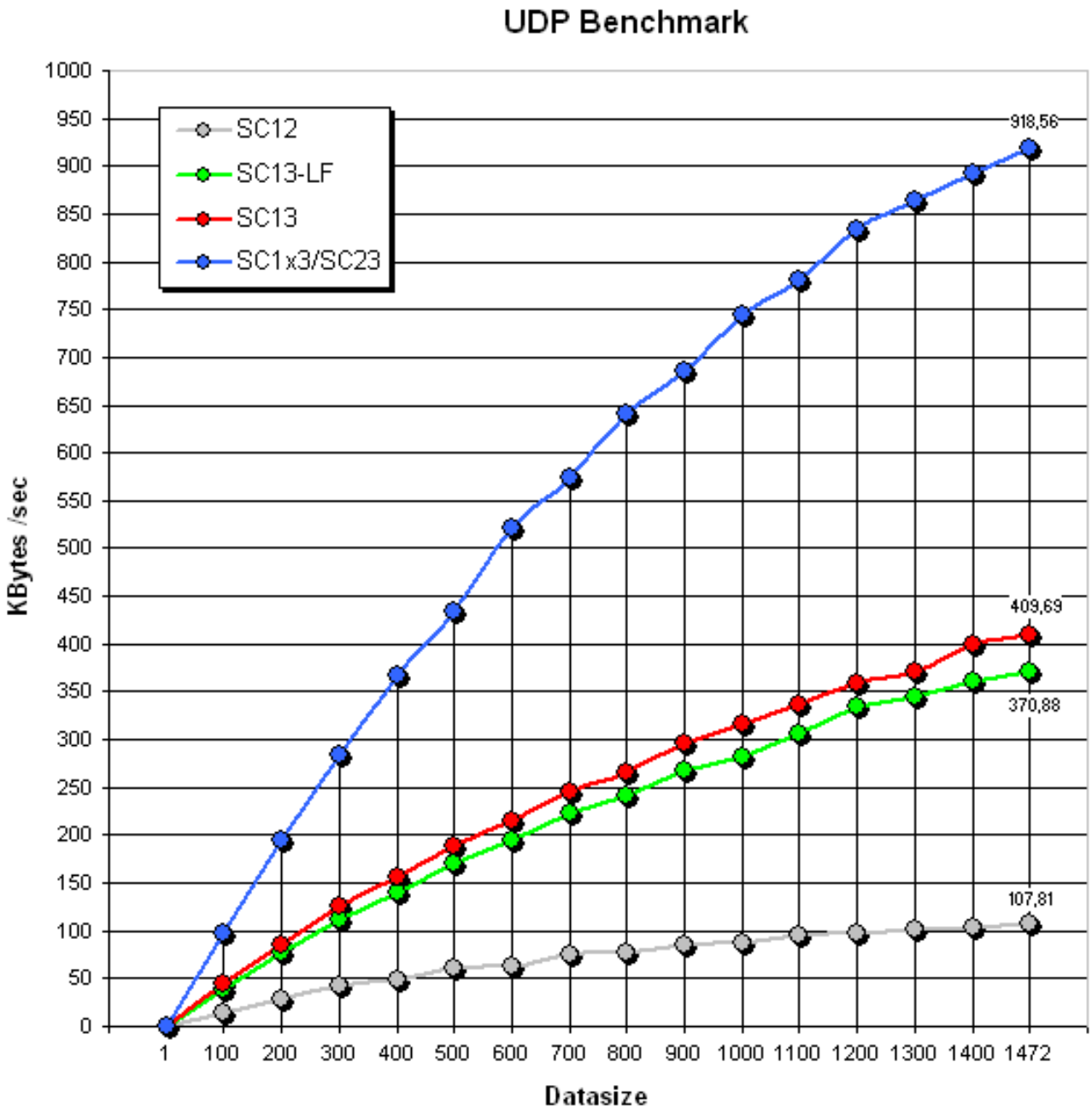
**UDP echo**

An UDP Echo client application, running on WinXP (Intel Pentium IV, 2,8 GHz) sends UDP datagrams of different data sizes to an IPC@CHIP® (SC12, SC13, SC13-LF, SC123/SC143, SC23) UDP Echo server application. The UDP server echoes the incoming datagramms back to the client. The client application measures the response time and calculates the amount of databytes in KBytes/sec, which the IPC@CHIP® is able to echo back to the client in dependency of the datagram size. The communication peers (WinXP PC and IPC@CHIP®) are connected with a "Twisted pair crossover cable" to avoid disturbtion by other network traffic.

This test was made under special test conditions. We cannot guarantee that the measured performance results are achievable on other test environment and conditions.

| | **UDP Echo<br>1472 bytes<br>KByte/s** |
|---|---|
| **SC12<br>10 Mbit/s, RTOS V1.25** | 107,81 |
| **SC13-LF<br>100 Mbit/s, RTOS V1.25** | 370,88 |
| **SC13<br>100 Mbit/s, RTOS V1.25** | 409,69 |

| SC123/SC143/SC23<br>100 Mbit/s, RTOS V1.20 | 918,56 |
|---|---|




End of document

## Programmable I/O Pins

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

**SC1x:**

```
Pin 1: RXD0/PIO7                  = RXD0
Pin 2: TXD0/PIO8                  = TXD0
Pin 3: CTS0/PIO9                  = Input pullup
Pin 4: RTS0/PIO10                 = Input pullup
Pin 5: TXD1/PIO11                 = TXD1
Pin 6: RXD1/PIO12                 = RXD1
Pin 7: TMROUT0/INT0/PIO13         = Input pulldown
Pin17: RESET/PFAIL/(LILED)[1]     = Input
Pin24: ALE/PCS0                   = Output, value High
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
```

1) LILED means:
SC12: Link/Traffic LED,
SC13: Traffic LED,
SC11: not available

**SC1x3:**

```
Pin C2:  TMRIN1/PIO0       = Input pullup
Pin B4:  TMROUT1/PIO1      = Input pullup
Pin K2:  PCS6/PIO2         = Input pullup
Pin K3:  PCS5/PIO3         = Input pullup
Pin H3:  PCS4/PIO4         = Input pullup
Pin P3:  PCS7/PIO5         = Input pullup
Pin B3:  ARDY/PIO6         = Input pullup
Pin B6:  DRQ2/PIO7         = Input pullup
Pin A4:  DRQ3/PIO8         = Input pullup
Pin E1:  PFI/PIO9          = Input pullup
Pin A5:  TMROUT0/PIO10     = Input pullup
Pin A2:  TMRIN0/PIO11      = Input pullup
Pin C7:  DRQ0/PIO12        = Input pullup
Pin D15: I2CDTA/PIO13      = Input pullup
Pin G1:  CANTXD1/PIO14     = Input pullup
Pin G2:  CANRXD1/PIO15     = Input pullup
Pin J2:  PIO16             = Input pullup
Pin D3:  HOLD/PIO17        = Input pullup
Pin C11: CTS3/SLVSEL/PIO18 = Input pullup
Pin B12: RTS3/SCK/PIO19    = Input pullup
Pin A10: RTS2/PIO20        = Input pullup
Pin B9:  CTS2/PIO21        = Input pullup
Pin A11: TXD2/PIO22        = Input pullup
Pin C10: RXD2/PIO23        = Input pullup
Pin L3:  PIO24             = Input pullup
Pin T6:  A23/PIO26         = Input pullup
Pin A12: TXD3/SDO/PIO27    = Input pullup
Pin B11: RXD3/SDI/PIO28    = Input pullup
Pin A6:  DRQ1/PIO29        = Input pullup
Pin A3:  INT5/PIO30        = Input pullup
Pin D16: I2CCLK/PIO31      = Input pullup
```

---

**SC23:**

```
Pin 31: TMRIN1/INT5/PIO0     = Input pullup
Pin 30: TMROUT1/INT3/PIO1    = Input pulldown
Pin 27: INT1/PIO2            = Input pulldown
Pin 32: PFI/PIO9             = Input pullup
Pin 29: TMROUT0/PIO10        = Input pullup
Pin 28: TMRIN0/PIO11         = Input pullup
Pin  3: CTS1/PIO12           = Input pullup
Pin 24: I2CDTA/PIO13         = Input pullup
Pin 12: SLVSEL#/CTS3/PIO18   = Input pullup
Pin 11: SCK/RTS3/PIO19       = Input pullup
Pin  8: CANTXD1/RTS2/PIO20   = Input pullup
Pin  7: CANRXD1/CTS2/PIO21   = Input pullup
Pin  5: TXD2/PIO22           = Input pullup
Pin  6: RXD2/PIO23           = Input pullup
Pin 10: SDO/TXD3/PIO27       = Input pullup
Pin  9: SDI/RXD3/PIO28       = Input pullup
Pin 25: I2CCLK/PIO31         = Input pullup
```

---

For further details about the IPC@CHIP® pin configuration, take a look to the dedicated hardware manual.

---

**Back to main index page**

---

End of document

---




# @CHIP-RTOS - Security notes

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 TCPIP network 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 @CHIP-RTOS 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 CHIP-RTOS Variants LARGE, MEDIUM, SMALL, TINY.
5. **Additional protection of a specified path**: Access is only, if the user authenticate himself by a defined **username** and **password**.

**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 CHIP-RTOS 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 @CHIP-RTOS 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 TCPIP network security

1. **Install System Server Connection Handlers** provides the possibility to generate IP- and/or Port - filters and forbid connections to FTP, WEB or Telnet
2. The BIOSINT API function **Suspend System Servers** allows you to Supend/Resume the FTP, Web and Telnet Server while runtime.
3. TCPIP API function **Register an IP callback filter function** allows the application programmer to install a filter callback function on every incoming IP packet.
4. TCPIP API function **Register an ARP callback filter function** allows the application programmer to install a filter callback function on every incoming ARP packet.

**Top of list**
**Index page**

---

End of document

---




# @CHIP-RTOS - Programming notes



## Introduction

Here are some useful notes for programming applications for the IPC@CHIP®. These notes contains general rules for programming DOS applications for the IPC@CHIP® and important information about internals of the @CHIP-RTOS which can prevent the programmers from making fatal errors.


- **Common notes for building IPC@CHIP® user applications**
- **Using RTOS API**
- **Programming CGI functions**
- **Configure the FTP server**
- **Usage of the TCP/IP API**
- **General notes for the usage of the DOS and @CHIP-RTOS int API calls**
- **Using Hardware API**
- **Using Fossil API**
- **Working with Float Data Types**
- **Configure PPP client or server**
- **SC1x3/SC23 Extended Memory and EXE File Format**


## Common notes for building IPC@CHIP® user applications

1. Limited Flash Write Cycles (of IPC@CHIP®s internal Flash):
   To open files on the flash disk drive A: for reading or writing, you can use the standard C functions fopen or open. We recommend the use of the fopen/fread/fwrite calls instead of open/read/write to reduce the flash write cycles, because these cycles are limited.
   For further information see the **FlashWriteCycles** document.

2. Compiler option settings:
   - 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)!

3. For inexperienced programmers, we recommend the use of the Large memory model for user applications. While Small (or Medium) memory models can offer more efficient use of memory and better performance than Large memory model, problems difficult to diagnose may occur when using the near data memory models such as:

   `. All pointer parameters passed to the @CHIP-RTOS API must be far pointers. Normally, the compiler will handle this conversion when interfacing to the Beck C-Library functions. But for functions that use a variable length argument list such as the `helper_printf`,

one can end up generating invalid code without any warnings offered by the compiler.

a. @CHIP-RTOS tasks **created** in a Small memory model program must have their stack memory declared in the program's data segment. (This requirement is due to the compiler's "SS equals DS" assumption used in near data memory models.) small memory model programmers who allocate their @CHIP-RTOS task stack space using the system memory **allocation** function, or declared as far data will observe some very strange program malfunctions and face tedious debugging.

b. If floating point **emulator** is used within @CHIP-RTOS tasks, small memory model should not be used due to the emulator's stack requirements.

4. No "exception handling" in Paradigm or Borland C 4.x or 5.x projects:
Users of Paradigm or Borland C-compilers should choose for their program project "No exceptions" at the "Target Expert" window. This setting saves Flash and RAM memory space.

5. Do not select "Test stack overflow" in the Paradigm or Borland C project compiler settings:
If "Test stack overflow" is selected a stack check is done at every function call. If stack overflow is recognized the program halts. This mechanism is not usable in applications which create additional @CHIP-RTOS **tasks** inside the application, because every task has its own stack and the compiler's runtime library stack functions are not aware of these additional stacks.

6. 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.

7. Before starting the first Turbo Pascal program at the IPC@CHIP® the **memopt 1** command must be executed (e.g. in the **AUTOEXEC.BAT**)

8. 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.

9. For C program development, the Beck C-Library should be used. This library provides efficient wrapper functions around the software interrupt API offered by the @CHIP-RTOS. Both the library source and binaries are available from the Beck Web site download area.

10. It is not advisable to design an IPC@CHIP® application as a set of several DOS programs for various reasons including:

`. It is usually more efficient to run several tasks (created with the **RTOS API**) inside of a single DOS program rather than running several DOS programs as tasks of the @CHIP-RTOS. The single program should require much less RAM and flash disk storage space compared to the memory requirements for a set of programs.

a. It can lead to high memory fragmentation and insufficient memory, if more than one or two DOS programs are running on the IPC@CHIP® when these programs are often terminated and restarted again.

11. It is not advisable to use in your application the Borland C-library `malloc` function with memory model Large. 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 fails, 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 what maximum memory is required by your `malloc` calls. It may be better to statically reserve the memory inside your application by declaring an unsigned character array (or the type you require) of the required size.

12. The @CHIP-RTOS allocates memory in the following manner:

- DOS programs are loaded into the lowest address free memory block of sufficient size.
- Memory blocks allocated inside of the @CHIP-RTOS or with **DOS service 0x48** are taken from the first free memory block of sufficient size, starting the search from the highest

heap RAM address.

So the largest free memory block in the system is usually located in the middle of the @CHIP-RTOS memory area. The shell command **mem** shows the state of the internal memory map.

13. Compressed executable files

    Compressing executable files with e.g. **pklite** or **upx** are useful to reduce the required diskspace. In case that the compressed applications are executed by a batch file the following behaviour should be considered:
    If **Batchmode 0** is set, the command shell task (MTSK) waits about 500 ms between executing each line of the batch file. In case of executing a large compressed application the execution of the next application may fail due to the large time (more than 500 ms) required for decompression of the executable in memory. The command shell prompts the error message **"Not enough memory to load program"** . To avoid this problem **Batchmode 1** should be used. It is also possible to insert an additional **delay command** at the batchfile between the lines of executing applications. A delay of 1 second ( **wait 1** ) should be sufficient.

**Top of list**
**Index page**

---

## Using RTOS API

1. Timer procedure's execution time must be kept as short as possible, because they are executed in the Kernel task. Long timer procedures block the kernel and other tasks of the @CHIP-RTOS. Do not use slow stack hungry C-library functions like `printf` inside of a timer procedure. This can cause a fatal stack overflow in the kernel task (stack size 1024 Bytes)

2. Timer procedure declaration with Paradigm or 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 an @CHIP-RTOS task with a default priority of 25. If the user application does not **yield**, lower priority **tasks** like the FTP server or the Web server will not work. In major loops within user applications, the programmer should insert **sleep calls** (or otherwise yield the CPU) if the FTP or Web servers' operation is desired during the execution 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 Paradigm or 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 the 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.

11. It is possible to **create tasks** with a time slicing value in the **taskdefblock structure.**. Please note that the kernel executes time slicing only between tasks which have the same priority. Other priority tasks are not affected.

**RTOS API**

**Top of list**
**Index page**

---

## Programming CGI functions

1. Avoid large loops inside of CGI functions. CGI functions must be executed as fast as possible. Long execution times in CGI functions block the Web server task, preventing response to other HTTP requests.

2. Declaring of CGI functions with Paradigm or 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 **Web server stacksize** in chip.ini up to 6144 Bytes, to prevent stack overflows, if Microsoft C-library functions like `sprintf` are used.

9. Dynamic HTML or text pages which are created inside a user CGI function should be as small as possible. The Web server inside must allocate a temporary buffer 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 Web server will need some memory for allocating temporary buffers for this page. How much memory should be left 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 the **Remove CGI page** API.

**CGI API**

**Top of list**
**Index page**

---

## Configure the FTP server

The default FTP idle timeout 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.

**Top of list**
**Index page**

---

## Usage of the TCP/IP API

1. Processing in a socket callback function (see **Register callback**) should be kept at a minimum to prevent stack overflows.

2. Declaring of socket callback functions with Paradigm or 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 TCP/IP stack of the IPC@CHIP® allocates memory for buffers smaller than 4096 bytes from a pre-allocated memory block. Larger buffers are allocated direct from the CHIP-RTOS. If user TCP/IP 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 in the IPC@CHIP®. The **mem command** shows the whole memory list of the IPC@CHIP® at runtime. The maximum amount of TCP/IP memory can be configured at chip.ini (see **Set TCP/IP memory size**). The application programmer can reduce or increase this size in chip.ini. With the API call **Get TCP/IP memory info** it is possible to control the TCP/IP memory usage at the application runtime.
The amount of memory required by the TCP/IP stack depends on the number of open sockets and the size and number of transported data packets. For user specific applications we recommended to test every used TCPIP application simultaneous and check how much memory from the pre-allocated block is used. This result should be used by the application engineer to set the **TCPIPMEM** entry of chip.ini to a sufficient value (+ a reserve of about 20%-30%).
With BIOS interrupt 0xA0 it is possible to install a user error handler function which will be called if

the memory limit is reached.

**See also:**

**TCP/IP API**
SC1x3/SC23 **IPv4/v6**Dual layer stack

**Top of list**
**Index page**

---

## General notes for the usage of the DOS and @CHIP-RTOS 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 **@CHIP-RTOS INT API**. If not, use the @CHIP-RTOS **BIOS execute shell command** INT call accessible through the `BIOS_Execute` C-library function. This call executes a shell command from inside the user application.

4. User error handler can be installed with the **BIOS_Install_Error_Handler** API. The user callback could then, for example, execute a **BIOS_Reboot** to reset the IPC@CHIP® on fatal errors.

5. Used RTOS software interrupts:

`0x00` - Reserved (@CHIP-RTOS Divide Overflow Handler)
`0x01` - Reserved (Debugger Trace Interrupt)
`0x02` - Reserved (Hardware Non-Maskable Interrupt (NMI))
`0x03` - Reserved (Debugger Breakpoint Interrupt)
`0x04` - Reserved (@CHIP-RTOS INTO Overflow Handler)
`0x05` - Reserved (@CHIP-RTOS Array Bounds Exception Handler)
`0x06` - Reserved (@CHIP-RTOS Invalid Opcode Exception Handler)
`0x07` - Reserved (@CHIP-RTOS ESC Opcode Exception Handler)
`0x08` - Reserved (Hardware, Timer #0 Handler)
`0x09` - Reserved
`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** (and Hardware UART #2 Handler)
`0x11` - **Biosint** (and Hardware UART #1 Handler)
`0x12` - Reserved (Hardware, Timer #1 Handler)
`0x13` - Reserved (Hardware, Timer #2 Handler)
`0x14` - **Fossil Interface** (and Hardware UART #0 Handler)
`0x15` - Reserved (Hardware INT5, UART #3 Handler)
`0x16` - **Biosint** (and Hardware, DMA #2 Handler)
`0x17` - Reserved (Hardware, DMA #3 Handler)
`0x18` - Reserved (Hardware, CAN Handler)
`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
0x34 thru 0x3E - Used by Borland (or Paradigm) floating-point emulator
0x51 - Floating Point **Emulator** (Paradigm Beck Edition)
0xA0 - **Several 'chip' related services**
0xA1 - **Hardware API** (HAL)
0xA2 - **Hardware API** (PFE)
0xAA - **I2C / SPI Interface**
0xAB - **CGI / SSI Interface**
0xAC - **TCP/IP API**
0xAD - **RTOS API**
0xAE - **Ethernet Packet Driver**
0xAF - Timer Interrupt, see **Set timer AF interval**
0xB0 - **External Disk API**
0xBF - **USB** and **CAN** API's (SC1x3/SC23 only)

6. Used Add-on driver software interrupts:

0x67 - Dynamic Library Service
0x6F - BCxxx device driver
0xB1 - **External Disk Driver**
0xB2 - **External Disk Driver** (SC1x3/SC23 only)
0xC0 - PPPoE API (SC1x3/SC23 only)
0xC1 - WLAN WL01 API (SC1x3/SC23 only)
0xC2 thru 0xCF - Reserved for future drivers

7. Software interrupts that can be freely used inside user applications:

0xD0 thru 0xDF - These vectors are available for user applications

8. External hardware interrupts are enabled (STI opcode) during execution of the following API software interrupts:

```
DOS ints 0x10, 0x14(Fossil), 0x16, 0x20, 0x21, @CHIP-RTOS int
0xA0, PFE int 0xA2, I2C int 0xAA, CGI int 0xAB, Pkt int 0xAE,
Extdisk int 0xB0, Extdisk user int 0xB1, CAN/USB int 0xBF
```

The state of the interrupt flag is not changed during the execution on the following API software interrupts (If interrupts are disabled before the API call, they are kept disabled during execution of the call. If interrupts are enabled before the API call, they get reenabled during execution of the call.):

```
TCPIP int 0xAC, RTOS int 0xAD
```

External hardware interrupts are disabled during execution of the following API software interrupts:

```
HAL int 0xA1
```

**BIOS ints**
**Int21h**

**Top of list**
**Index page**

# Using Hardware API

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 normal HW API user ISR. For this purpose install a RTX user ISR, see **Install ISR**.

3. The latency time of the user ISR (from generation of an interrupt until first line of code inside the user ISR) depends on the used IPC@CHIP®, see **Performance comparison** .

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.

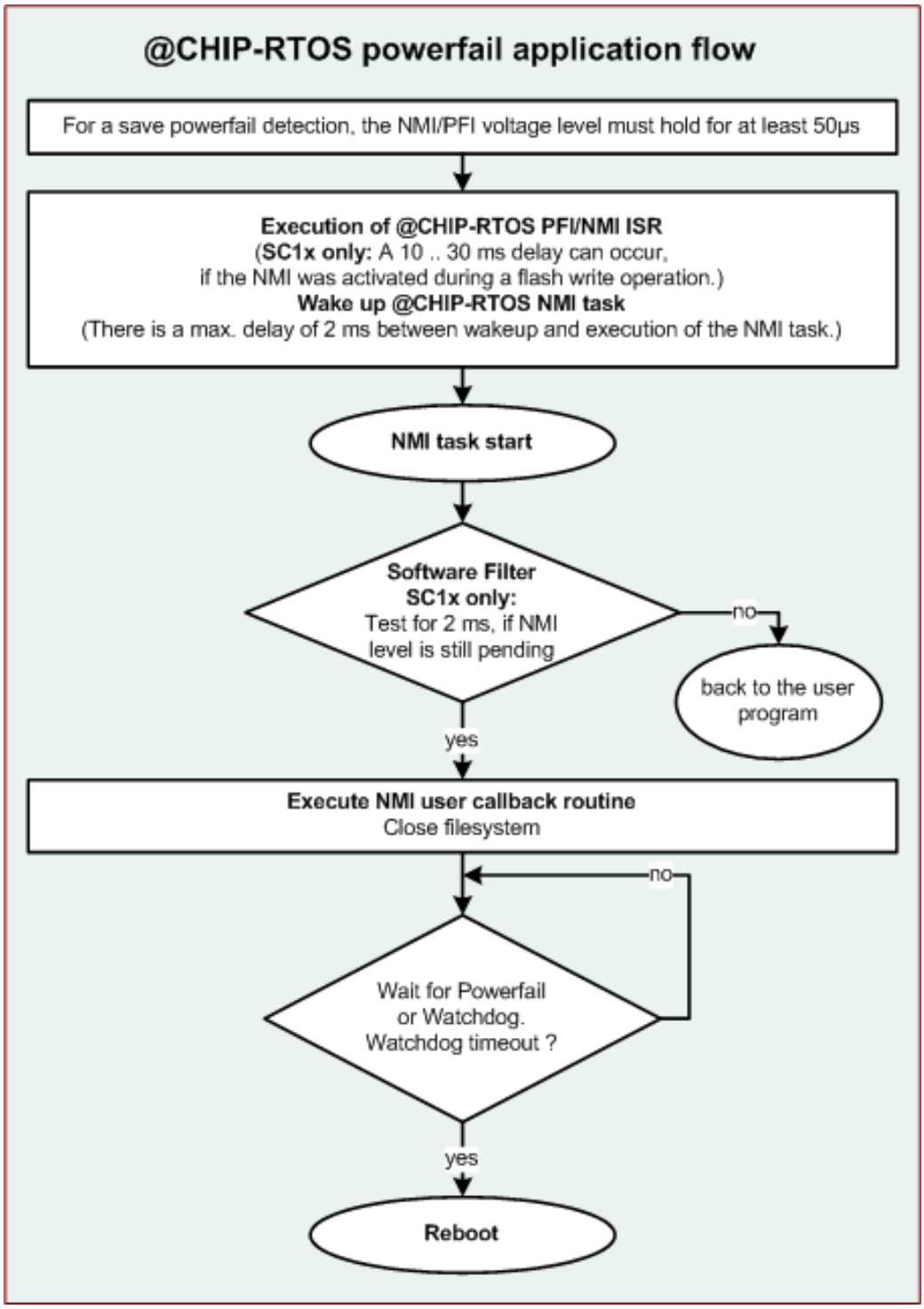
5. **Usage of DMA0/INT5 and DMA1/INT6:**
   DMA0 and INT5 are served inside the @CHIP-RTOS with a single shared interrupt handler. (Likewise for DMA1 and INT6.)
   If a user application has activated **serial EXT DMA mode** and also wants to use external interrupt INT5, the DMA interrupt event is higher priority. (And again, likewise for COM DMA mode and INT6.)
   This means that if a DMA0/1 interrupt occurs simultaneous with an external INT5/6 interrupt, the installed user handler for the external interrupt is not called.

   See **Hardware API**

6. Usage of NMI/Power fail detection:

   The NMI function of the multifunction pin 17 (RESET/NMI/LINK_LED) of the IPC@CHIP® SC11/SC12/SC13 is for power fail purposes only. It is not possible to use NMI as a "normal" interrupt pin like e.g. INT0 for generating interrupts. It can only be used as described in the IPC@CHIP® hardware documentation.

   The flowchart below describes how the @CHIP-RTOS handles an incoming NMI interrupt.




**See also:**

[Install Interrupt Service Routine]
[Init non-volatile data]
[Save non-volatile data]

[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 to use external DMA, you have to disable the **serial DMA receive mode**. Otherwise the DMA receive mode is recommended.

3. Since SC12 @CHIP-RTOS 1.02B XON/XOFF mode is available also with the serial DMA receive mode in use.
   Please note: Because of the internal functionality of DMA it is not possible to immediately detect an XON or XOFF character from the connected peer. It is possible that an overrun situation at the peer (e.g. GSM modem) can occur. Nevertheless, we enable this mode because some GSM modems (any??) support only XON/XOFF as serial flow control mechanism.

4. The default serial receiver queue size is 1024 bytes. If the default DMA receive mode is used, it is advised to increase the receiver queue size in **Chip.ini** up to a minimum value of 2048 bytes to prevent a possible buffer overrun (even if hardware handshake is used). This can only happen with the default queue size of 1024 bytes if the user doesn't call the **Fossil API read block function** frequently enough. If the application programmer does not increase this buffer size up to the recommended value, they should call the **Fossil API read block function** with sufficient buffer size in the CX-Register to flush the internal buffers and prevent a receive buffer overrun.

5. **Serial ports, running with IRQ receive mode:**
   The two serial ports of the SC1x IPC@CHIP® have no hardware FIFO buffer. Only one incoming character can be stored directly by the ports. (SC1x3/SC23 IPC@CHIP® serial ports have a four deep receiver FIFO.) As a consequence of this hardware, it is possible that incoming characters get lost due to delayed receiver interrupt execution (see **Operating mode of serial ports**). For example, writing a flash sector (which occurs when writing a file to drive A:) disables all interrupt execution in SC1x systems for about 15 ms. (For SC1x3/SC23, the interrupts are not masked throughout the sector write period.) If the serial port is configured with a baud rate of 9600, incoming characters can be lost during this time.

   Loss or delayed execution of serial receiver interrupts depends on the number and the execution frequency of all enabled interrupts in the IPC@CHIP® system. It depends also on execution times of user interrupt service functions and the duration of interrupt masking periods (CLI / STI instruction sequences).

6. For a given serial port the fossil functions are not all reentrant. However you may, for example, call the transmitter services from task A and receiver services from task B without problems. The port must first be configured before this concurrent operation of transmitter and receiver is permissible. For different serial ports the fossil functions are fully reentrant, including the port configuration functions. The behavior of the fossil API when a serial port device is used in a reentrant fashion (e.g. two tasks attempt to call a transmitter service at the same time for the same device) is controlled within the API using a semaphore, one semaphore for send and another for receive.

   **Fossil API**

**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 Paradigm Beck Edition or 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".

   **CAUTION:** The floating-point emulators are implemented using a set of software interrupts. Consequenctly, only a single DOS program may use the floating-point

emulators at any one time. Otherwise one of your DOS programs is likely to crash during floating-point usage.

2. Using the floating-point math emulation inside @CHIP-RTOS tasks with the Paradigm Beck Edition is relatively simple. Three things must be considered.

    a. Do not use memory model small or medium.
    b. Assure that the task's stack includes offset zero. (See discussion below how this can be achieved.)
    c. Call `_fpinit()` from your @CHIP-RTOS task before performing any floating-point operations to initialize the task's stack for use by the floating-point emulator.

    If for some reason your code must be compatible with both the Borland 5.02 and Paradigm Beck Edition compilers, or if you have an older Borland 5.02 program which you want to leave as is, you can also use the more complicated floating-point initialization required for Borland 5.02 described below. The Paradigm Beck Edition compiler tolerates this form of floating-point initialization as well.

3. The Borland/Paradigm Math Emulation libraries are not made for usage in a multitasking system like the @CHIP-RTOS. If you want to use float data types in tasks other than your main (DOS) task, please pay attention to the following work around solution:

    a. For floating point emulator usage, the current stack (the stack of the task) must have offset 0. One way to achieve this is to allocate the task stack memory directly from the @CHIP-RTOS memory pool using **int21h 0x48** (or the corresponding `helper_alloc_rtos_mem` CLIB function).

    ```
    unsigned int task_stacksegment =
                helper_alloc_rtos_mem(STACK_SIZE);
    unsigned char * task_stackptr =
                MK_FP(task_stacksegment, 0);

    task_defblock.stackptr = &task_stackptr[STACK_SIZE];
    task_defblock.stacksize = STACK_SIZE;
    ```

    Another way is to normalize a pointer for the task stack:

    ```
    #define PARAGRAPH_SIZE (16)
    unsigned char task_stack[STACK_SIZE+PARAGRAPH_SIZE];
    unsigned int huge *fp_huge_stack;
    unsigned char far *fp_stack;

    fp_huge_stack = (unsigned int huge *)
        &task_stack[PARAGRAPH_SIZE-sizeof(unsigned int)];
    fp_huge_stack++; //Compiler normalizes pointer
    fp_stack = (unsigned char far *)
        MK_FP(FP_SEG(fp_huge_stack), 0);
    task_defblock.stackptr = (unsigned int far *)
        &fp_stack[STACK_SIZE];
    task_defblock.stacksize = STACK_SIZE;
    ```

    Notes:
    - For SC1x3/SC23 IPC@CHIP® systems, the PARAGRAPH_SIZE is 256 instead of 16.
    - The CGI callbacks execute on the Web server's stack, which satisfies this "offset 0 present" requirement.

    b. Before executing floating point arithmetic inside tasks, the current stack must be pre-initialized for math emulation.

    Within Paradigm Beck Edition this is done by calling the `_fpinit()` function. Example for a task function:

    ```
    void huge my_task(void)
    {
    ```

```
    float a;
    _fpinit();
    // Ready for floating point arithmetic
}
```

Within Borland 5.02 this can be achieved with the following code.
Example for a task function:

```
void my_emu1st(void)
{
    asm{
        // Set end of FPU stack
        mov word ptr ss:[0x2f],0x0065
        // Set start of FPU stack
        mov word ptr ss:[0x31],0x0125
        mov word ptr ss:[0x27],0 // no protected mode
        mov byte ptr ss:[0x26],0 // no 8087
    }
}

void huge my_task(void)
{
    float a;
    my_emu1st();
    _fpreset();
    // Ready for floating point arithmetic
}
```

If you are interested in more details, take a look at the Paradigm or Borland 5.02 RTL sources (`fpinit.asm`). We provide an example for using floating point arithmetic in separate tasks or within CGI callbacks.

4. Using Math emulation with Borland C++ 5.02 (also for older versions of Borland C) produces a conflict between the NMI interrupt of the @CHIP-RTOS (Interrupt Vector 0x02) and the Borland floating exception handling. Borland C math emulation also uses `INT 2` for generating floating point exceptions. Our internal NMI interrupt service function (normally called at power fail case) will no longer execute during the runtime of the user's application. We have provided a solution which moves the Borland exception function from INT 2 to another interrupt vector. We modified the Borland RTL source (`fpinit.asm`) and include the modifications directly into the application project. An example is provided in our Internet download area.

   Note: For Paradigm Beck Edition compiler there is no conflict between NMI vector and the floating point exception handler. The Paradigm compiler uses interrupt 0x51 for the floating point exception.

**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® 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 initialization **structure**. But the `idletimeout` and `modemctrl` detection can 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 can 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 the PPP client initialization **structure**). Most modems use RTS/CTS flow control, if they get the AT command AT\Q3.

3. The COM and EXT port of the IPC@CHIP® 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).

4. If the option PPP_IPCP_PROTOCOL (VJ TCP/IP header compression) is set (see **PPPCLIENT_SET_OPTIONS**), it is possible that the FTP server of the IPC@CHIP® is not usable via the PPP interface. This was noticed at PPP sessions connected to a Linux PPP peer.

**PPP**

**Top of list**
**Index page**

---

## SC1x3/SC23 Extended Memory and EXE File Format

### Extended Memory

The SC1x3/SC23 computers operate with a 256 byte *paragraph* size instead of the conventional 16 byte paragraphs on traditional DOS computers. This means that translation between segmented addresses and linear addresses on the SC1x3/SC23 systems would be done as:

```
typedef unsigned long DWORD ;
void far *ptr ;
DWORD linear_addr = ((DWORD) FP_SEG(ptr) << 8 )
                            + FP_OFF(ptr) ;
```

as opposed to the traditional DOS x86 computer's similar calculation:

```
DWORD linear_addr = ((DWORD) FP_SEG(ptr) << 4 )
                            + FP_OFF(ptr) ;
```

This provides the SC1x3/SC23 with an address space of 2**24 bytes, or 16 MByte which overcomes the standard DOS limitation of 1 MByte address space (2**20).

### EXE File Format

The *Paradigm Beck IPC Edition* compiler's linker takes into consideration this 256 byte paragraph size as it allocates the segments for an extended mode program. The resulting EXE executable file header is tagged with an 'MX' designation to distinguish it from a standard 'MZ' DOS program EXE. This allows the @Chip-RTOS to detect if an EXE has been properly linked for the respective target system type. The error message

```
"file is not a valid executable"
```

is issued to the console if an attempt is made to use an EXE type not suited for the particular system.

Below is the EXE header data structure, as viewed by the @CHIP-RTOS. Structure members which receive special handling by the @CHIP-RTOS are highlighted.

```
typedef struct {
   union {
         unsigned char Byte[2];
         unsigned int Word ;
```

```
        } Signature ;
    unsigned int nBytesLastPage;         // 02, 03
    unsigned int nPages;                 // 04, 05
    unsigned int nRelocations;           // 06, 07
    unsigned int nParagraphsHeader ;   // 08, 09
    unsigned int minParagraphs ;       // 0xA
    unsigned int maxParagraphs ;       // 0xC
    unsigned int initialSS;              // 0xE
    unsigned int initialSP;              // 0x10
    unsigned int chksum;                 // 0x12
    unsigned int initialIP;              // 0x14
    unsigned int initialCS;              // 0x16
    unsigned int startOfRelocationtable; // 0x18
    unsigned int nOverlays ;           // 1A
    unsigned long   Time_Tag ;         // 0x1C
    unsigned int _Pad1;                  // 0x20
    unsigned int RN_Tag ;              // 0x22
    unsigned int _Pad2 ;                 // 0x24
    unsigned long ActualFixupEntries ;   // 0x26
    unsigned int _Pad3 ;                 // 0x2A through 0x2B
    unsigned long ParadigmTimeTag ;    // 0x2C
} FILE_HEADER;

#define RN_SIGNATURE    (((WORD)'N' << 8) | 'R')
```

The SC1x3/SC23 @CHIP-RTOS uses special handling for the following members of the EXE file header.

**Signature** - These two bytes must be 'M', 'X' in order to launch on the SC1x3/SC23. The 'M', 'Z' DOS programs will not launch with one exception: If the `nOverlays` member is set to magic number 0xABCD, then the program is considered to be a legal TINY program and it will be launched provided that the stack space indicated by `initialSS` and `initialSP` members fits within the program's single 64 Kbyte segment.

**nParagraphsHeader** - This is the number of 16 byte DOS paragraphs which the EXE file header occupies. Note that the *paragraph size* used here is 16 bytes for either extended mode (SC1x3/SC23) or normal DOS (SC1x).

**minParagraphs** - This is the minimum number of paragraphs that the @CHIP-RTOS will allocate as the program is loaded. Here hardware paragraph size is used, which is 16 bytes for SC1x and 256 bytes for the extended mode SC1x3/SC23 computers. (Exception: The flat memory 'M', 'Z' TINY DOS programs which run on either target use 16 byte paragraph size here.)

**maxParagraphs** - This is the maximum number of paragraphs that the program needs. This value is not used by the @CHIP-RTOS. It is mentioned here only to state that hardware paragraph size is used, which is 16 bytes for SC1x and 256 bytes for the extended mode SC1x3/SC23 computers.

**Time_Tag** - Place where TIMETAG tool outputs a 32 bit build timestamp, seconds past year 1969. When the *Paradigm Beck IPC Edition* compiler linker is used, this field is ignored.

**RN_Tag** - Special indicator set by *Paradigm Beck IPC Edition* compiler linker to indicate that the program time-tag should be taken from the `ParadigmTimeTag` location, alternative to the old Beck convention at offset 0x1C. This field is set to `RN_SIGNATURE` by the *Paradigm Beck IPC Edition* linker.

**nOverlays** - Set to 0xABCD in order to force execution of a 'M', 'Z' type DOS program on SC1x3/SC23. (Note: One way this header field can be set to 0xABCD is with the -x option on the TIMETAG tool distributed with Debug@Chip. Only specially prepared flat memory model (TINY) standard DOS programs can be expected to execute on the SC1x3/SC23. The PROBE.EXE used by the debuggers and the CHIPEDIT.EXE example program are such flat memory model programs, which can be executed on both SC1x and SC1x3/SC23 targets. For any normal SC1x3/SC23 program, the *Paradigm Beck IPC Edition* compiler's linker must be used. Creating the cross-target flat memory programs is not straight forward and is beyond the scope of the support users can expect from Beck. These type programs are mentioned here only to make this @CHIP-RTOS EXE

header documentation complete.)

**ParadigmTimeTag** - This build timestamp is automatically set by the *Paradigm Beck IPC Edition* compiler linker. This 32 bit build timestamp (seconds past year 1969) is used by the debuggers to correlate target code with symbol tables.

**Top of list**
**Index page**

---

End of document

---

# @CHIP-RTOS - Boot Flow chart

IPC@CHIP® Documentation **Index**

## Introduction

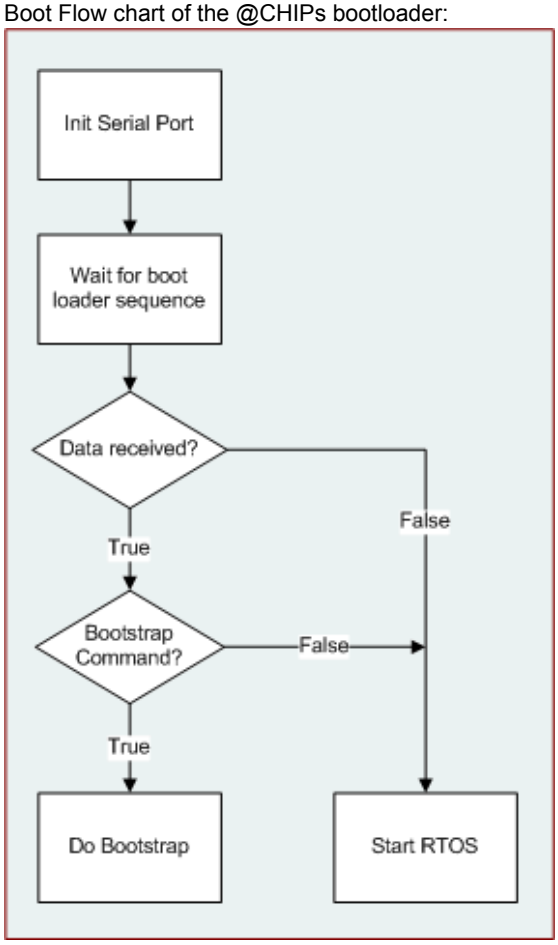
This page describes the boot process and its error handling of the IPC@CHIP®. Also a list of all possible errors during boot up is attached.


- **Boot Flow Chart**
- **Possible errors during the boot process**

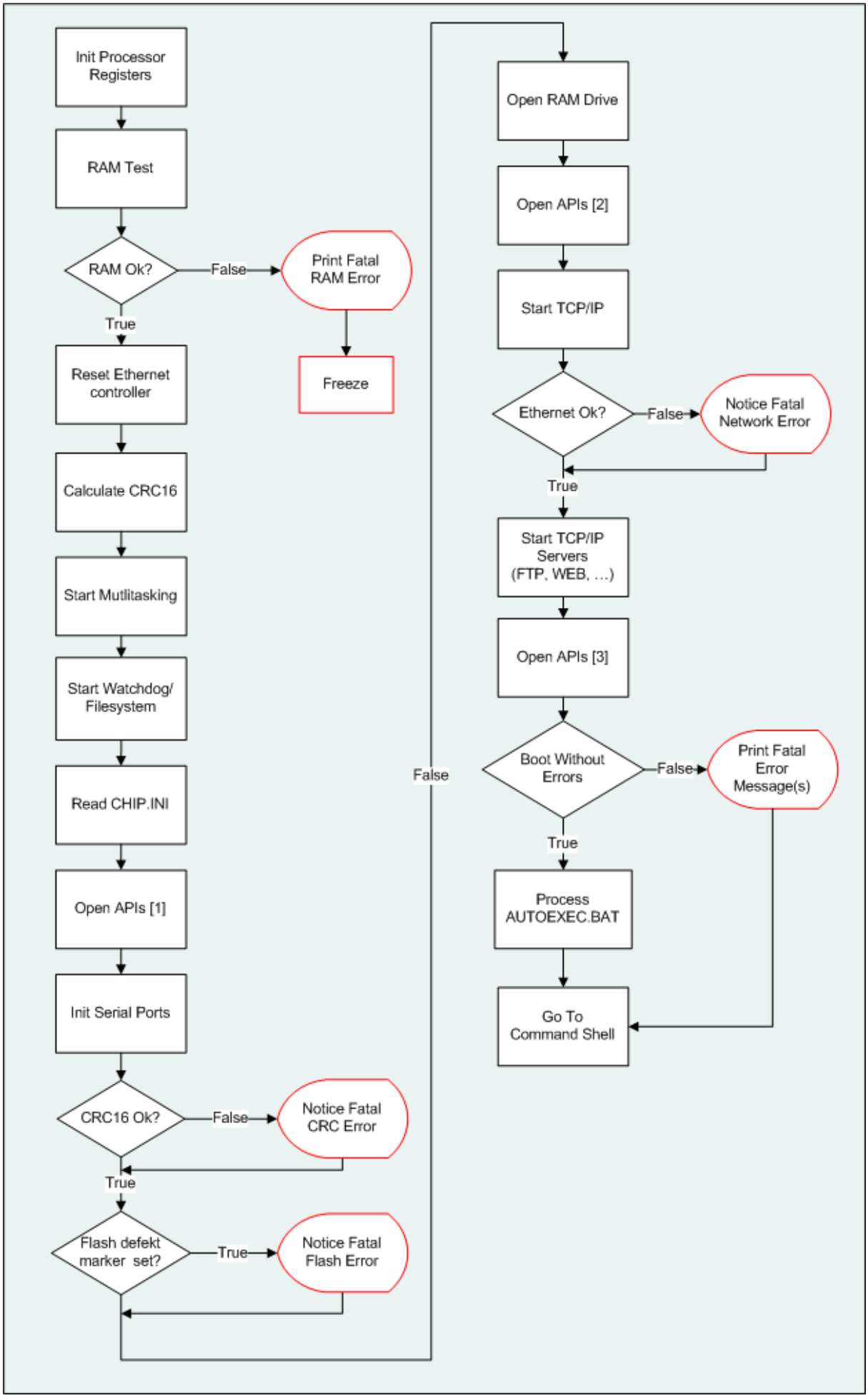

## Boot Flow Chart

The boot process consists of two steps. In the first step the bootloader will be executed. It checks on the serial EXT port if an @CHIP-RTOS update request comes in (from CHIPTOOL). If not, the bootloader goes to the second step by jumping into the @CHIP-RTOS startup code. Below there are the flow charts of step one (bootloader) and step two (@CHIP-RTOS)
The errors in the diagrams will be described in the next point (**Possible Error while the boot process**)

1. Boot Flow chart of the @CHIPs bootloader:




2. Boot Flow chart of the @CHIP-RTOS:




Top of list
Index page

## Possible errors during the boot process

During the boot process several errors could occur. All possible errors are mentioned in the list below. They prevent start of the **AUTOEXEC.BAT**, so your application will not be started start automaticaly (to prevent fatal errors at runtime). The error which occured will be printed on the command shell.

1. RAM Error:
   On startup of the @CHIP-RTOS a RAM test will be performed. If it fails, the @CHIP prints the error message "RAM ERROR" on the COM port (using 19200 Baud and 8N1, regardless the entry in the CHIP.INI). The @CHIP-RTOS freezes after the error message output (does not boot!).
2. Checksum Error:
   During the boot process a checksum over the complete @CHIP-RTOS image in the flash will be generated. This checksum must be equal to the checksum which was written with the @CHIP-RTOS update (in a special sector on the flash). If the checksum is not equal, the error message "Fatal BIOS checksum error" will be shown on the serial console. The system will boot, but the execution of the AUTOEXEC.BAT will be prevented.
3. Flash Error:
   Every flash write access of the @CHIP-RTOS will be verified. If the flash access fails, an internal flag will be set, to mark the sector which should be written as defect. On the next reboot the error message "Flash error at sector: xx" will be shown on the serial console. The system will boot, but the execution of the AUTOEXEC.BAT will be prevented.
4. Network Error:
   During the init process of the TCP/IP it could be possible that a network error occurs (mostly it is a hardware defect). In this case the message "Fatal Network Error" will be printed on the serial console of the IPC@CHIP®. The system will boot, but the execution of the AUTOEXEC.BAT will be prevented.

**Top of list**
**Index page**

---

End of document

---

# @CHIP-RTOS - Command Processor

## COMMAND

Command processor
Interprets the commands in AUTOEXEC.BAT and those issued at the console.

### File system commands

- CD directory
- COPY filename1 filename2
- DEL filename
- DIR filename
- FORMAT drive [/C:n] [/E] [/R:n] [/Q]
- FTLSTAT
- FTLRECL
- MD directory
- TYPE filename
- RD directory
- REN filename1 filename2

### Hardware commands

- ADR addressline
- ALE 0/1
- IW address
- IB address
- OW address value
- OB address value
- RW segment:offset
- RB segment:offset
- WW segment:offset value
- WB segment:offset value
- MEMDMP segment:offset size
- PCS chipselect
- PIO [pionumber mode]

### TCP/IP commands

- IP address
- NETMASK mask
- GATEWAY address
- IPCFG
- IPV6 index address
- IPV6CFG

- IPETH
- IKE
- TCPIPMEM
- DHCP 0/1
- FTP 0/1
- TFTP 0/1
- PING address
- CLOSETELNET
- CGISTAT
- WEBSTAT

### RTOS commands

- MEM
- TASKS
- UTASKS
- ERRORS
- VER

### Other commands

- BATCHMODE 0/1
- MEMOPT 0/1
- CON device [device] ...
- WAIT secs
- XTRANS port s/r filename
- REBOOT
- HELP
- SERVPROG [exe_filename [password]]

---

## CD directory
## Change the current working directory.

Changes the current working directory.

### Example

```
cd temp
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## COPY filename1 filename2
## Copy a file.

Copy a file. The two file specifiers must be complete file names.

Wildcards such as * or ? are not allowed.

### Comments

This command will fail if the resulting overall full path length of the new file exceeds a maximum (see below). This full path includes the drive letter and back slashes in the count, e.g.:

A:\TEMP\MY_FILE.TXT

Each of the string length specifications here do not include in the count the required final string terminating NIL character.

### SC1x Comments

The over all maximum path length is 147 characters. The names in each of the path segments must be 8.3 maximum, DOS fashion.

### SC1x3/SC23 Comments

The over all maximum path length is 259 characters. The names in each segment of the full path are limited to 255 characters maximum.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

Top of list
Index page

---

## DEL filename
## Delete a file

Delete a file or all files that match the wildcard.

### Comments

Files which have the *Read-only* attribute set cannot be deleted.

### Example

```
del *.dat
```

### Related Topics

DIR command
DOS file attributes API

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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.

The file's attributes are shown with three letter designations. The letter is replaced by a dash, '-', when the respective attribute is not set. These three attributes are, from left to right:

A - Archive flag
R - Read-only file
S - System file

The user is free to use the A and S flags as they please. These flags play no role inside the @Chip-RTOS file system operation.

The read-only attribute will prevent files from being **deleted** so long as this flag is set.

There also is a *Hidden* file attribute flag. The DIR command does not list files that have the hidden attribute flag set.

Note that C language applications can control these attribute flags with the `_dos_setfileattr()` standard C-library function.

### Example

```
dir *.exe
```

### Related Topics

**DEL** command
DOS file **attributes** API

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## FORMAT drive [/C:n] [/E] [/R:n] [/Q]
## Format a disk drive.

Format a drive.

**All information on the drive will be lost !**

### Comments

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

The cluster size parameter /C: is optional. This parameter cannot be applied to the C: **RAM** drive, which automatically selects its own cluster size appropriate for the FAT12 file allocation table which is used. For the other

drives, the specified cluster size must be one of following legal values:

1, 2, 4, 8, 16, 32, 64

The default cluster size for the internal A: drive is 2 sectors per cluster. (The file system's sector size is 512 bytes.) The default cluster size for external B: and RAM C: drives vary depending on the volume size.

If the /E parameter is specified, the data area will be filled with null-data. This option is not supported on the RAM C: drive.

With parameter /R you can specify the number of root directory entries. Note that the actual number of root entries must be a multiple of 16, so your specified number of root entries will be rounded up if necessary to the nearest multiple of 16.

If the quiet mode parameter /Q is specified, the confirmation check is skipped. Specify this if you like to call this command with the BIOS_execute() API function.

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

## Example

```
FORMAT A: /C:2 /E
FORMAT B: /C:4
FORMAT B: /R:256
```

## SC12 Comments

On drive A: flash disk, there may result 16 more root directory entries than requested due to internal cluster alignment requirements.

## SC1x Comments

When /C: option is used to specify the sectors per cluster for the external B: drive, care must be taken that the resulting number of clusters does not exceed the FAT16 limit of 65524 data clusters. Otherwise the formatting attempt will fail.

The default numbers of root directory entries are:
32 entries on A: flash drive
512 entries on B: external drive
16 entries on C: RAM drive

B: drive volumes over 2 Gbyte will be truncated to 2 Gbyte so that they can be managed with a FAT16 format file allocation table.

## SC1x3/SC23 Comments

The SC1x3/SC23 @Chip-RTOS file system supports FAT32, allowing large B: or D: drive volumes to be used.

The default numbers of root directory entries are:
64 entries on A: flash drive
512 entries on B:/D: external drives
32 entries on C: RAM drive

Note that long file names eat up additional directory entries according to name length as:

number_of_extra_directory_entries = (strlen(filename) + 12) / 13 ;

The 8.3 DOS style file names require no additional directory entries on the A: flash drive and C: RAM disk. This is also true for the external B:/D: drive provided that the file name was all upper case.

The /R option for root directory entry count has no meaning for FAT32 volumes and will be ignored. (A FAT32 root directory has no particular limit on number of entries.)

An additional command line option for the external B:/D: drive can be used to explicitly specify either FAT32 with

```
FORMAT B: /F:32
```

or

```
FORMAT B: /F:16
```

for FAT16 format. Depending on a disk's sector count, either of these options may fail if an appropriate *sectors per cluster* count cannot be found, or if a *sectors per cluster* count specified with the /C: option does not suite the specified FAT16/FAT32 format. The /F: option is valid only for B: and D: drive.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.15 | V1.00 |

Top of list
Index page

---

## FTLSTAT
## Display state of flash translation layer.

Displays the state of the flash translation layer an drive A:, like free, dirty and valid sectors.

### Comments

The flash translation layer provides the logic for the file system to access small virtual sectors (512 Byte) from larger physical flash sectors (64 kB). This is necessary, because only complete physical sectors can be erased. This layer also assures a wear leveling, so that data gets spread across the media.

The FTLSTAT command shows the state of the managed small virtual sectors. Valid sectors are sectors that have valid content. Free sectors are sectors that have all bits in the erased state. Dirty sectors are sectors that have been discarded or replaced. Dirty sectors are automatically reclaimed, if the free sector count reachs a low number. Reclamation means that all valid sectors in a physical block that has dirty sectors are copied to another physical block, before the whole physical block is erased to recover the dirty sectors.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## FTLRECL
## Reclaim sectors in flash translation layer.

Reclaims all dirty sectors in the flash translation layer on drive A:.

### Comments

The filesystem does an auto reclaim, if the free sectors reach a specific number. A manual operated reclaim is not necessary.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

## MD directory
## Creates a directory.

Creates a directory.

### Comments

This command will fail if the resulting overall full path length exceeds a maximum (see below). This full path includes the drive letter and back slashes in the count, e.g.:

A:\TEMP\

Each of the string length specifications here do not include in the count the required final string terminating NIL character.

### Example

```
md temp
```

### SC1x Comments

The over all maximum path length is 147 characters. The names in each of the path segments must be 8.3 maximum, DOS fashion.

### SC1x3/SC23 Comments

The over all maximum path length is 259 characters. The names in each segment of the full path are limited to 255 characters maximum.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

Top of list
Index page

## TYPE filename
## Type a file.

Show contents of a file on the console.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## RD directory
## Removes a directory.

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

### Example

```
rd temp
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## REN filename1 filename2
## Rename a file.

Rename a file or directory.

### Comments

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

Both files must reside in the same directory.

Files or directories which are in use by a task or program cannot be renamed.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.16 | V1.00 |

Top of list
Index page

---

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

**SC1x:**

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

**SC1x3:**

Enables the non-multiplexed address bus pins (A23).
The command expects only one parameter: 23=enable A23

### Example

```
ADR 0
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

---

## ALE 0/1
## Enable/disable ALE pin.

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

### Example

```
ALE 1
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | n/a |

Top of list
Index page

---

## IW address
## Input word.

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

### Example

```
IW 600
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## IB address
## Input byte.

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

### Example

```
IB 600
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## OW address value
## Output word.

Perform a 16 bit output on a given I/O address with given data.
The address and the data are hexadecimal. The address is the first parameter followed by data.

### Example

```
OW 600 F
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## OB address value
## Output byte.

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

### Example

```
OB 600 F
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## RW segment:offset
## Read word from memory.

Perform a 16 bit read from a given segment:offset memory address.
The address and the result are hexadecimal.

### Example

```
RW 8000:100
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | n/a |

Top of list
Index page

---

## RB segment:offset
## Read byte from memory.

Perform an 8 bit read from a given segment:offset memory address.
The address and the result are hexadecimal.

### Example

```
RB 8000:100
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | n/a |

Top of list
Index page

---

## WW segment:offset value
## Write word to memory.

Perform a 16 bit write on a given segment:offset memory address with given data.
The address and the data are hexadecimal. The address is the first
parameter followed by data.

### Example

```
WW 8000:100 1A2B
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | n/a |

Top of list
Index page

---

## WB segment:offset value
## Write byte to memory.

Perform an 8 bit write on a given segment:offset memory address with given data.
The address and the data are hexadecimal. The address is the first
parameter followed by data.

### Example

```
WB 8000:1000 1A
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | n/a |

Top of list
Index page

---

## MEMDMP segment:offset size
## Dump memory area.

Dump a memory area on the console from a given segment:offset memory address.
The address is hexadecimal. The size is decimal. The results are show in hexadecimal and ascii.

### Example

```
MEMDMP 8000:100 256
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | n/a |

Top of list
Index page

---

## PCS chipselect
## Enable chip select.

Enables a chip select line.
The command expects only one parameter: the chip select line.

**SC1x:**
Valid arguments are 0, 1, 2, 3, 5 or 6.

**SC1x3:**
Valid arguments are 0, 1, 2, 3, 4, 5, 6, 7.

### Example

```
PCS 6
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

---

## PIO [pionumber mode]
## Enable and show PIO pins.

**SC1x:**

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

**SC1x3:**

Enables the programmable PIO pins (PIO0-24, 26-31).
The command expects two parameters: PIO MODE

PIO: PIO number (0-24, 26-31)
MODE: PIO mode
- 1,2 = Input with pullup
- 4 = Output value = High
- 5 = Output value = Low

**SC23:**

Enables the programmable PIO pins (PIO0-2, 9-13, 18-23, 27-28, 31).
The command expects two parameters: PIO MODE

PIO: PIO number (0-2, 9-13, 18-23, 27-28, 31)
MODE: PIO mode
- 1,2 = Input with pullup/down (PIO1/PIO2=pulldown, all other=pullup)
- 4 = Output value = High
- 5 = Output value = Low

When no command line argument is given, the PIO state is shown.

### Example

```
PIO 3 5  = PIO3 Output low
PIO      = Shows PIO states
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## IP address
## Sets the IPv4 address of the Ethernet interface.

Sets the IPv4 address of the internal Ethernet interface.

### Comments

This command modifies the information stored in `A:\chip.ini`.

The DHCP option is also switched off.

The new address is not used until after a **IPETH** command or a restart of the system.

Use the **IPCFG** command to verify your entry before restarting the system.

### Example

```
IP 195.243.140.85
```

### Related Topics

IPv4 address **initial value**
Set an **IPv6** address
**IPCFG** - IPv4 configuration display command
Set IPv4 Address **API** function
**PPP** server initial IP address
Initial **DHCP** setting

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## NETMASK mask
## Set the network mask for IPv4 addressing of the Ethernet interface.

Sets the subnet mask for IPv4 addressing of the internal Ethernet interface.

### Comments

This command modifies the information stored in `A:\chip.ini`.

The DHCP option is also switched off.

The new subnet mask is not used until after a **IPETH** command or a restart of the system.

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
Set IPv6 address
Initial DHCP setting
Set IP subnet mask API function

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

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

Sets the IPv4 address of the default gateway to use.

### Comments

This command modifies the information stored in `A:\chip.ini`.

The DHCP option is also switched off.

The new gateway address is not used until after a IPETH command or a restart of the system.

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

## IPCFG
## Display current IPv4 configuration

This command display current IPv4 configuration of all installed TCP/IP device interfaces. The fields shown for each installed TCP/IP device interface are:

Device: Interface name
Type: ETH(Ethernet), LPK(Internal loopback), PPP or Unknown
Idx: Internal index number of the device interface
IP address
Network mask
MAC address: (If Ethernet type)
Gateway: Default gateway, if one is set

The current link state and operating mode of the Ethernet controller are displayed with the IPv4 address configuration of the default Ethernet device.

### Example

```
IPCFG
```

### Related Topics

**IPV6CFG** - IPv6 configuration
IP - Set IPv4 address **command**

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## IPV6 index address
## Set an IPv6 address of the Ethernet interface.

Set an IPv6 address entry for the internal Ethernet interface.

### Comments

Command syntax: `IPV6 AddressIndex Address` (see examples below)

It is possible to configure up to 5 IPv6 addresses (index 0 through 4). This command modifies the **information** stored in `A:\chip.ini`.

The new address is not used until after a **IPETH** command or a restart of the system.

The prefix length is always required (e.g. the "/64" in examples below).

The **IPV6CFG** command can be used to verify your entry before restarting the system.

### Example

```
IPv6 0 3ffe:ffff:0:f101::1/64
IPv6 1 4ffe:ffff:0:f101::1/64
```

```
IPv6 4 2ffe:ffff:0:f101::1/64
```

## Related Topics

**IPV6CFG** - IPv6 configuration display command
IPV6 address **initial value**
IPV6 **Stateless autoconfiguration**
Set **IPv4** address command

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# IPV6CFG
# Display current IPv6 address configuration

This command display the current IPv6 address configuration of all installed TCP/IP device interfaces. The fields shown for each installed TCP/IP device interface are:

Device: Interface name
Type: ETH(Ethernet), LPK(Internal loopback), PPP or Unknown
Idx: Internal index number of the device interface
IPv6 addresses
MAC address (if Ethernet type)

Fixed configured addresses are listed with index numbers 0-4. Addresses configured by "Stateless auto configuration" are listed with index number 5 - 9. The current link state and operating mode of the Ethernet controller are displayed with the IPv6 address configuration of the default Ethernet device.

## Example

```
IPV6CFG
```

## Related Topics

Set an **IPv6** address
IPV6 **address** initial value
IPV6 **Stateless autoconfiguration**
**IPCFG** - IPv4 configuration
Overview **IPv4/v6** - Dual layer stack

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# IPETH

## Restart the Ethernet interface

Restart/Reconfigure the internal Ethernet interface, e.g. after changing the IP configuration,
without rebooting the system.
Opt. parameter 'N': Reconfigures the IP settings with physical init of the ethernet controller
Opt. parameter 'A': Reconfigure ethernet with an AutoIP address(available since SC1x RTOS V1.20 and SC1x3 RTOS V1.05).

### 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 error code 237 signals that a Ethernet configuration was already in progress.

Please note: If the specified number of **DHCP trials** is larger than default 3, the command will not wait for completion of the DHCP process.
The AutoIP procedure is also usable by the chip.ini entry **AUTOIP**.

### Example

```
ipeth    ; Reconfigures the IP settings and initializes the ethernet controller physically
ipeth N  ; Reconfigures the IP settings of the ethernet device
ipeth A  ; Reconfigures the IP settings of the ethernet device by using the AutoIP procedure
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## IKE
## Internet Key Exchange (IKE) status.

Lists IP Security and IKE status information to console. A list of established Security Associations (SA) is provided.

### Comments

START is an optional parameter to this command which can be used to start up the IKE task if this has not already been **started**. Starting the IKE task enables use of the **IP Security**.

The error **codes** listed here are provided to help resolve problems with either IP security startup processing or IKE negotiation with a peer.

When IKE negotiations are failing in phase 2 (Quick mode), the Security Association (SA) parameters that we propose (initiator role) or can accept (responder role) from the respective peer are displayed for diagnostic purposes. When we are the responder and the peer is the initiator, the peer's SA proposals will also be listed (this recording is limited to the first 6 transforms received). Therefore more Quick mode diagnostic information is available here when the peer side initiates the IKE.

### Example

```
IKE START
```

### Related Topics

**IPSEC** ENABLE in CHIP.INI

IPsec_Start() API

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

---

## TCPIPMEM
## Display TCP/IP memory usage

Displays TCP/IP memory usage. This command shows the maximum reserved memory for the TCP/IP stack and the current TCP/IP stack memory used.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Enables or disables the use of DHCP for the internal Ethernet interface to obtain an IP configuration.

### Comments

DHCP is an abbreviation for "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

Initial DHCP setting

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

## FTP 0/1
## Enable/Disable FTP.

Enables (1) or disables (0) the FTP server. Additionally a resp. entry is added to the `chip.ini` configuration file to preserve the FTP server's running status after a reboot.
Note that `chip.ini` is not changed if the command is issued from within a **batch file**.

### Example

```
FTP 1
```

### Related Topics

Initial **FTP** ENABLE setting

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## TFTP 0/1
## Enable/Disable TFTP

Enables or disables file transfers via TFTP server.
0 disables the server, 1 enables TFTP file transfer.

### Comments

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

### Example

```
TFTP 1
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| on request | on request | on request | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## PING address
## 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.
DNS names are supported.

### Comments

DNS queries consume a relative large part of the main task's relative small stack.

### Example

```
PING 192.168.200.10
PING www.beck-ipc.com
```

### Related Topics

**PING_OPEN** API function

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## CLOSETELNET
## Close current telnet session.

This function will finish the current Telnet session.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## CGISTAT
## List Installed CGI handlers.

This function will list all installed CGI handlers.

### Related Topics

**CGI_INSTALL** API function

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## WEBSTAT
## Show the current settings of the Web server

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

See WEB `config` for the available chip.ini entries for the Web server.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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.

### SC1x Comments

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

### SC1x3/SC23 Comments

One sector (256 bytes) is added for memory management.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## 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.

## Comments

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. At the first call of TASKS, the timer interrupt routine of the RTOS is exchanged by a version for the task monitor. First after 10 seconds, a repeated TASKS command will return usable results.

The command shows no task stack size for DOS applications, 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 by monitored.

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

The listed task state is only a one moment snapshot. The task state bit field is a 16 bit hexadecimal value defined as follows:

| | |
|---|---|
| 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 |
| Bit8-15 | internal use only |

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

Very high priority:

```
AMXK      prio= 00      Kernel task
NMI       prio= 01      Powerfail task
TCPT      prio= 04      TCP/IP timer task
ETH0      prio= 05      Ethernet receiver task
```

Normal:

```
PPPS      prio= 06      PPP server
PPPC      prio= 06      PPP client
CFGS      prio= 07      UDP config server
TELN      prio= 11      Telnet server
SSHD      prio= 11      SSH server (daemon)
SSHx      prio= 11      SSH server (session), x=0,1,2,...
MTSK      prio= 12      Console task (command shell)
```

Low priority:

```
WEBS      prio= 41      Web server
FTPS      prio= 41      FTP server
SSLW      prio= 41      SSL web server
```

## Example

```
Sample output:
ID 1094 cnt  3515  MTSK prio= 12  stack=3000 used=35% state=0000
ID 1606 cnt    81  ETH0 prio=  5  stack=2048 used=41% state=0004
ID  256 cnt  4568  AMXK prio=  0
ID 2374 cnt  1072  WEBS prio= 41  stack=2048 used=24% state=0081
ID 2886 cnt   100  DOS1 prio= 25  stack= n/a used=n/a state=0040
ID 3142 cnt   157  DOS2 prio= 25  stack= n/a used=n/a state=0081
ID 1862 cnt    24  CFGS prio=  7  stack=1400 used=28% state=0004
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## UTASKS
## Disables the Task Monitor.

Disables the Task Monitor which was installed using **TASKS** command.

If you do not need the Task Monitor anymore, you should disable it using this command because the Task Monitor has a performance penalty.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## ERRORS
## Display list of all detected errors.

Displays a list of all detected errors.

```
Message                         Meaning

FATAL BIOS CHECKSUM ERROR       Corrupted RTOS code at flash memory detected
FLASH ERROR AT SECTOR           Flash write error has occurred
FATAL NETWORK ERROR,            Fatal error at the network controller detected
INVALID OPCODE ERROR AT TASK    CPU has generated an invalid opcode interrupt
REACHED TCP/IP MEMORY LIMIT     TCPIP memory pool is exhausted
TCP/IP MEMORY ALLOC ERROR       TCPIP memory allocation failed
TCP/IP INTERNAL FATAL ERROR     Fatal internal TCP/IP error
LOW MEMORY ERROR                An allocation from the CHIP-RTOS memory pool failed
CORRUPT MEMORY ERROR            The RTOS memory list is corrupted/invalid
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## VER
## @CHIP-RTOS Version.

Output the IPC@CHIP® serial number, @CHIP-RTOS version and build date.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## BATCHMODE 0/1
## 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## 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.

### Example

```
MEMOPT 0
```

### SC12 Comments

With SC12 @CHIP-RTOS version 0.67, the default for MEMOPT is disabled. Earlier versions had this feature enabled but this resulted in errors with `malloc()`.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## CON device [device] ...
## Direct console I/O.

Define what device is used as the console. Possible devices are COM, EXT, TELNET, SSH (SC1x3/SC23 only) or USER. Multiple devices are possible.

### Comments

This setting is only valid until the next reboot.

### Example

```
CON COM
CON EXT
CON USER
CON COM TELNET SSH
```

### Related Topics

Default input device **STDIN** initial value
Default output device **STDOUT** initial value

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.07 | V1.00 |

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
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## XTRANS port s/r filename
## File transfer: Send/Receive file with Xmodem.

**SC1x:**
Send/Receive file with XMODEM/CRC protocol. Possible devices are COM, EXT and USER.

**SC1x3:**
Send/Receive file with XMODEM/CRC protocol. Possible devices are COM, EXT, SER2, SER3 and USER.

**SC23:**
Send/Receive file with XMODEM/CRC protocol. Possible devices are COM, SER2, SER3 and USER.

### Example

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Displays a list of all available console commands.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# SERVPROG [exe_filename [password]]
# Install Service Program.

**SC1x3/SC23:**

This command installs a program into a hidden reserved area of system Flash memory outside the scope of the file system. This program will then be automatically started at system start-up prior to **AUTOEXEC.BAT** file processing.

The maximum executable file size is 32742 bytes. Compression with the UPX_Beck.exe tool (distributed with Beck Postmake and Debug@Chip) is recommended.

Note that this special Service Program interacts with the system in the same manner as any program launched from the AUTOEXEC.BAT. When **BATCH_MODE** '1' is used, the system console thread will hang for the [BATCH] **EXECTIMEOUT** period unless the application Service Program calls the **BIOS_Batch_Continue()** API.

**WARNING**

Use of this feature is not generally advised by Beck! To do so, you may reach a configuration under which the system cannot be started due to this program. You then have no command prompt with which you can remove or change this program. Note that the Ctrl-C use at system startup which can abort or bypass the AUTOEXEC.BAT ***does not*** prevent this Service Program from executing. Only thoroughly tested DOS programs should be installed here!

Examples of this command follow. The first command line parameter is the EXE file name. An optional second parameter is the password.

**To check if Service Program exists:**

Enter command with no arguments to check if a Service Program is currently present in Flash memory.

```
SERVPROG
```

**To load a Service Program:**

Following command loads MyProg.EXE as the Service Program, provided that no password protection had been established.

```
SERVPROG MyProg.EXE
```

**To load a Service Program with password:**

Append a password to restrict further changes to the Service Program storage area. Further use of the SERVPROG command will require this same password in order to perform modifications to this Service Program store. This example will store MyProg.EXE as the Service Program provided that the current password is either non-existant or it matches the entered password. Note: Passwords are handled in a case sensitive manner.

```
SERVPROG MyProg.EXE MySecretPassword
```

**To remove Service Program:**

Using **remove** as the file name is a special case. When this string appears as the filename, any existing Service Program will be removed (provided that any required password protection is satisfied). After program removal, any previously used password is no longer required on subsequent SERVPROG commands. In this manner, by removing and installing the program again, the password can be changed.

This example assumes password protection has been established. Otherwise no password would be required here.

```
SERVPROG **remove** MySecretPassword
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

Top of list
Index page

End of document




# @CHIP-RTOS - AUTOEXEC.BAT



## AUTOEXEC.BAT

The AUTOEXEC.BAT file is a special batch file that is executed after system boot. It can be used to automatically start programs or process any other shell commands right after RTOS has started up. The AUTOEXEC.BAT is not processed if any boot error occures. (See **Boot flow chart**)

It is also possible to stop the execution of the AUTOEXEC.BAT manually by pressing CTRL + C (via a terminal) while the system starts up. The @CHIP-RTOS will in this case ask whether the file should be executed or not.
This behaviour can be disabled via a **CHIP.INI entry**.

## What is a batch file?

A batch file is a plain-text file listing a batch of RTOS shell commands that will be executed by the command shell. The shell commands must have the same format as if they were put in on the shell prompt, e.g. you can omit the file extension .EXE when calling executable programs.

Each line of the batch file may contain one command. The commands will be processed line-by-line in the given order. Depending on the batchmode, the shell will wait with the execution of the next line until the previous command has finished or has allowed further batch processing. The batchmode can be modified using a **CHIP.INI entry** or with the corresponding **shell command**.

By definition batch files have the file extension .BAT. This is needed for the command shell to identify a file as a batch file. Besides the AUTOEXEC.BAT there can be many other batch files on your IPC@CHIP® that you can call by hand. This is especially useful to create shortcuts for commands which need a lot of arguments.

**Top of list**
**Index page**

End of document




# @CHIP-RTOS - CHIP.INI Documentation



## CHIP.INI

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. The CHIP.INI file has the following structure:

```
[SECTION]
Item1=value1
Item2=value2
```

Sections have to be unique. Lines inside a section can be commented out with a leading semicolon.

**[STDIO]**
STDIN
STDOUT
FOCUS
FOCUSKEY
CTRL_C
PRIO

**[DEVICE]**
FILESHARING
POWERSAVE
PFI_ENABLE
NAME

**[BATCH]**
BATCHMODE
EXECTIMEOUT

**[TIMER]**
1C
AF
RTI

**[DOSLOADER]**
MEMGAP

**[RAMDRIVE]**
SIZE
SEGMENT
WS

**[IP]**
ADDRESS
NETMASK
GATEWAY
DHCP
HOSTNAME_OPT
CLIENT_ID
DHCP_TRIALS
AUTOIP
TCPIPMEM
TCP_PRIO
ETH_PRIO
ETH_MODE
ETH_PHY
ETH_ENABLE
MTU
ARPENTRIES
ARPSMART
FORWARDING

**[IPv6]**
ENABLE
ADDRESSx
STATELESS_AUTOCONF

**[IPSEC]**
ENABLE
POLICY_FILE
PRESHARED_KEY
IKE_PRIORITY

**[TFTP]**
TFTPPORT
PRIO

**[WEB]**
ENABLE
PRIO
MAINPAGE
MAINPAGE_POST
TEMPPATH
DRIVE
ROOTDIR
MAXCGIENTRIES
WEBSERVERSTACK
HTTPPORT
POST_SIZE
HEADER_SIZE
HEADER_REJECT
CONNECTIONS
UPLOAD
USER0
PASSWORD0
SECURE
SEC_URLx
SEC_USERx
SEC_PASSWORDx
SSL_ENABLE
SSL_PRIO
SSL_DRIVE

**[PPPSERVER]**
ENABLE
PRIO
IPv6_USE
MODEMTRACE
COMPORT
ADDRESS
REMOTEADDRESS
NETMASK
GATEWAY
AUTH
MTU
IDLETIME
FLOWCTRL
MODEM
USERx
PASSWORDx
BAUD
INITCMDx
INITANSWERx
INITTIMEOUTx
INITRETRIESx
MODEMCTRL
CTRLTIME
CTRLCMDx
CTRLANSWERx
CTRLTIMEOUTx
CTRLRETRIES
CMDMODE

[TELNET]
ENABLE
PRIO
TELNETPORT
TIMEOUT
LOGINDELAY
LOGINRETRIES
USERx
PASSWORDx

[SSH]
ENABLE
SSHPORT
PRIO
PRIO_KEX
RSA_KEYFILE
DSA_KEYFILE
USERx
PASSWORDx
USERx_KEYFILE
LOCAL_FORWARD
REMOTE_FORWARD
FORWARD_ALL
REKEY_DATA
REKEY_TIME

IKE_BUF_SIZE
IKE_CACERTx
IKE_CLICACERTx
IKE_LOCALCERT
IKE_KEYFILE
IKE_PH1_AGGRESSIVE
IKE_PH2_PFS
NAT
NAT_TIMEOUT
NAT_LINGER

[FTP]
ENABLE
PRIO
CMDPORT
LOGINDELAY
TIMEOUT
USERx
PASSWORDx
ACCESSRIGHTx
DRIVEx
ROOTDIRx

SSL_ROOTDIR
SSL_HTTPPORT
SSL_CONNECTIONS
SSL_VERSION
SSL_CACERTx
SSL_CLICACERTx
SSL_SERVERCERT
SSL_KEYFILE
SSL_EPHFILE

[DNS]
NAME_SERVER1
NAME_SERVER2
CACHE_SIZE

[TRACE]
FLASHWRITE
INTNOTSUPP

[UDPCFG]
LEVEL
PRIO
IPCFG_PASSWORD
FLASH_PASSWORD

[USB]
MAX_POWER_SUPPLY
BULK_PER_FRAME

HANGUPDELAY
HANGUPCMDx
HANGUPANSWERx
HANGUPTIMEOUTx
HANGEUPRETRIESx
CONNECTMSGx
CONNECTANSWERx
CONNECTTIMEOUTx

[PPPCLIENT]
ENABLE
PRIO

[SERIAL]
EXT_DMA
COM_DMA
SER2_DMA
SER3_DMA
SEND_DMA
SEND_DMA2
EXT_RECVQUEUE
EXT_SENDQUEUE
COM_RECVQUEUE
COM_SENDQUEUE
SER2_RECVQUEUE
SER2_SENDQUEUE
SER3_RECVQUEUE
SER3_SENDQUEUE
COM_BAUD
EXT_BAUD

---

# [STDIO]
# STDIN=Define standard input device

Define your device for standard input. You can define several devices simultaneously. By default, both COM and TELNET are used.

## SC1x Comments

Valid devices are COM, EXT and TELNET.

## SC1x3 Comments

Valid devices are COM, EXT, TELNET and SSH. The SSH enables the pseudo terminal established with a SSH session.

## SC23 Comments

Valid devices are COM, TELNET and SSH. The SSH enables the pseudo terminal established with a SSH session.

## Example

The following example defines both COM and TELNET for stdin.

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.07 | V1.00 |

Top of list
Index page

---

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

Define devices used for standard output. You can define several devices simultaneously. By default, both COM and TELNET are used.

### SC1x Comments

Valid devices are COM, EXT and TELNET.

### SC1x3 Comments

Valid devices are COM, EXT, TELNET and SSH. The SSH enables the pseudo terminal established with a SSH session.

### SC23 Comments

Valid devices are COM, TELNET and SSH. The SSH enables the pseudo terminal established with a SSH session.

### Example

The following example defines both COM and TELNET for stdout.

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.07 | V1.00 |

Top of list
Index page

---

## [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 user's 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.

### Related Topics

FOCUS KEY configuration

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [STDIO]
## FOCUSKEY=Key

Defines the key that switches 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 pressing Ctrl-F keys on the console will then cycle the stdio between the three modes:

Stdio: User
Stdio: Shell
Stdio: Both

The new mode is shown on the console.

Key Range: 0..254

If the key is set to zero, then no focus key is defined and the switching of stdio is disabled.

Note:
The focus key code is filtered out by the system, and will not be visible to either the command shell or a DOS executable. This key code is therefore not usable by either the command shell or a DOS executable.

### Related Topics

Initial **FOCUS** configuration

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Disable/enable the possibility to prevent processing of the **AUTOEXEC.BAT** via CTRL + C key.
The following example disables the ctrl-c control.

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

By default, CTRL_C is enabled.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [STDIO]
## PRIO=prio

Defines the task priority of the shell task (MTSK).

Allowed value are 2-127. Default is 12.

Example:

```
[STDIO]
PRIO=20
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.15 | V1.00 |

Top of list
Index page

---

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

Defines the IPv4 address of the internal Ethernet interface, 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 or `ADDRESS=0.0.0.0` , the IP address will be set to `1.1.1.1.` This special behaviour is required to enable UDP communication between the UDP config server and a requesting CHIPTOOL application for setting a valid IP configuration.

### Related Topics

**IP** command line entry of IPv4 address
Set IPv4 Address **API** function
**IPv6** address initial value

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

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

Defines the IPv4 subnet mask of the internal Ethernet interface of IPC@CHIP®, 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 IPv4 subnet mask **API** function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [IP]
# GATEWAY=Gateway IP Address

Setting the default IPv4 gateway.

## Comments

Example: `GATEWAY=195.243.140.65`

The TCP/IP stack of the IPC@CHIP® supports only one valid default gateway for all device interfaces: Ethernet and PPP Interface.
So see also **PPP Server GATEWAY** if you are using PPP.

We provide some additional API functions for modifying the default gateway:

- Interrupt 0xAC, **Service 0x80**: add a default gateway
- Interrupt 0xAC, **Service 0x81**: delete default gateway
- Interrupt 0xAC, **Service 0x82**: get default gateway

Important:
As a valid chip.ini gateway entry at this section [IP], the TCP/IP stack accepts only a gateway ip address, which matches to the net address of the internal Ethernet device ip configuration.
If the user adds an own device driver interface (e.g. second Ethernet on SC1x3) with a new gateway setting, he has to set this gateway inside of his application by calling **AddDefaultGatewayEx**.

## Related Topics

CHIP.INI entry **PPPSERVER GATEWAY**
**GATEWAY** command line
Set IPv4 gateway **API** function
**ADD_DEFAULT_GATEWAY** API function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

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

Set to 1 if DHCP client should be used to get the IP configuration for the internal Ethernet interface 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## [IP]
## HOSTNAME_OPT=0/1 Insert device name at DHCP requests as option 12

If set to 1, the IPC@CHIP® insert at its DHCP request its hostname as option 12. This is the name defined at Device name.
If the additional entry CLIENT_ID is not defined, the device name string is also inserted at the DHCP request as client identifier option 61.

### Comments

By default this feature is disabled.

### Related Topics

DHCP client identifier option 61

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.03 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [IP]
# FORWARDING=0/1 Enables/disables ip forwarding

If set to 1, the TCP/IP stack of the IPC@CHIP® forwards ip packets between all network interfaces. Using forwarding the IPC@CHIP® acts as router between two networks. By example if an ip packet arrives on eth0 and the destination network address is equal to that one of eth1 the packet will be forwarded to eth1.
If defined as 0, ip forwarding is disabled for all interfaces.

## Comments

This option is enabled by default.

## Supported since or modified in @CHIP-RTOS version

| | | | | |
|---|---|---|---|---|
| SC12 | SC13 | SC11 | SC1x3 | SC23 |
| V1.25 | V1.25 | V1.25 | V1.20 | V1.20 |

Top of list
Index page

---

# [IP]
# CLIENT_ID=Insert client identifier string at DHCP requests as option 61

If this entry is specified, the IPC@CHIP® insert at its DHCP request the specified string as DHCP option 61.
If this entry is not specified but the entry **HOSTNAME_OPT** is yet defined, the IPC@CHIP® inserts here its **Device name**.
If CLIENT_ID is set to the reserved word "MAC_ADDR", the IPC@CHIP® builds a 7 Byte long client identifier option consisting of:

1 Byte Hardware type, with value 1 for Ethernet
6 Bytes physical Ethernet address.

Example: `CLIENT_ID=MAC_ADDR`

## Comments

By default this feature is disabled.
Maximum name size: 20 characters

## Related Topics

**DHCP hostname option 12**

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

Top of list
Index page

---

## [IP]
## DHCP_TRIALS=Number of DHCP configuration trials

Specify the number of DHCP configuration trials.

### Comments

The IPC@CHIP will try to obtain an IP configuration via DHCP several times. The number of trials can be configured with this option. Valid values are between 3 and 128. If you specify a value of 0, the configuration via DHCP will be retried endlessly (Not on SC1x!). Note that the configuration via **AutoIP** will be implicitely disabled in this case.

There are time intervals between each DHCP configuration attempt. At the first three DHCP trials, the DHCP client works with fixed time intervals.
The first three time intervals of the SC1x are 3, 3 and 6 seconds.
The first three time intervals of the SC1x3/SC23 are 3, 3 and 3 seconds.
After 3 DHCP configuration trials, the time interval is calculated by the term "1 << (trialNumber - 1)". E.g. Trial number 4 has a time interval of 8 seconds.
Time intervals are generally limited to 32 seconds.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

Top of list
Index page

---

## [IP]
## AUTOIP=0/1

If AUTOIP = 1 and DHCP = 1, the (first) Ethernet interface will be configured by the AutoIP algorithm, if the DHCP configuration fails.

### Comments

Default is 1. The AutoIP procedure configures the Ethernet interface with a random IP address between 169.254.1.0 and 169.254.254.255. After selecting a random IP, it executes an address collision detection by sending ARP probes. The procedure repeats with new random IP addresses, until no collision can be detected.
It is also possible to call the AUTO IP procedure "by Hand" with the command **IPETH A**.

Note that if the **DHCP_TRIALS** parameter is set to 0, DHCP configuration will never reach the failed state and thus the configuration via AutoIP will not be started.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | n/a | V1.05 | V1.00 |

Top of list
Index page

---

## [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.

Example: `TCPIPMEM=60`

**SC1x:**

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 @CHIP-RTOS configured without PPP capability (server or client) 98 kBytes when @CHIP-RTOS configured with PPP capability

**SC1x3/SC23:**

Valid Range: Between 60 kBytes and 1000 kBytes (An out of range value for TCPIPMEM will be set to closest of these limit values.)

Default value: 140 kBytes

### Comments

The TCP/IP API function 0x78, **GET_MEMORY_INFO**, reports the current used memory of the TCP/IP stack.

### SC12 Comments

Since SC12 @CHIP-RTOS version 1.02B we allow configuring a maximum value of 160 kBytes, because some application programmers may require more then the old limit of 132 kByte.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.06 | V1.00 |

Top of list
Index page

---

# [IP]
# TCP_PRIO=prio

Defines the task priority of the TCP timer task (TCPT).

Allowed value are 2-127. Default is 4.

Example:

```
[IP]
TCP_PRIO=10
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.15 | V1.00 |

Top of list
Index page

---

# [IP]
# ETH_PRIO=prio

Defines the task priority of the Ethernet receiver task (ETH0).

Allowed value are 2-127. Default is 5.

Example:

```
[IP]
ETH_PRIO=11
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | n/a | V1.15 | V1.00 |

Top of list
Index page

---

# [IP]
# ETH_MODE=Select operating mode of the internal Ethernet device

The internal Ethernet controller is able to operate in different operating modes. Select the following operating modes of the internal Ethernet device:

0: Auto negotiation, Default mode
1: Fixed 010 MBit/s, Half duplex
2: Fixed 010 MBit/s, Full duplex
3: Fixed 100 MBit/s, Half duplex
4: Fixed 100 MBit/s, Full duplex

Example: `ETH_MODE=1`

## Comments

For avoiding possible configuration errors at different networks, we recommend the usage of the auto negotiation mode.

## Related Topics

**Detect Ethernet link state** API function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [IP]
# ETH_PHY=Configure Ethernet phy

This entry (an 8 bit hex value) provides a (possibly required) alternative configuration of the Ethernet PHY.
**For experts only!**
We strongly recommend that you leave the default settings (0x22) unchanged. Change the default settings only after having contacted us (support@beck-ipc.com).
**Bit 7: Select cable type**
0: UTP(100 Ohm) cable type (Default)
1: STP(150 Ohm) cable type
**Bit 6: Receive Level Adjust**
0: Normal (default)
1: Receive Squelch levels reduced by 4.5dB
**Bit 5-2: Transmit Output Level Adjust:**
4 Bit value for adjusting the transmit output levels from approx. -14% to +16% in steps of 2%.
Default value for Bit5-2: 1000b (0%)
**Bit1-0: Transmitter Rise/Fall time**
2 Bit value for changing the adjustment range from -0.25ns to +0.5ns in steps of 0.25ns.
Default value for Bit1-0: 10b.

Example for changing the cable type from UTP to STP: ETH_PHY=0xA2

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | V1.00 | n/a | n/a | n/a |

Top of list
Index page

---

# [IP]
# ETH_ENABLE=0/1

Defines whether the internal Ethernet controller of the IPC@CHIP® is started or not. If ETH_ENABLE=0 the Ethernet device is not started. By default the Ethernet device is enabled.

## Comments

If ETH_ENABLE=0, the RTOS doesn't initialize the Ethernet controller at boot time.
Since SC1x3 RTOS 1.05 initialization of the Ethernet device can be done later by executing the command **IPETH**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [IP]
# MTU=Bytes

Sets the Maximum Transfer Unit of the internal Ethernet interface.

```
[IP]
MTU=1300
```

By default, MTU is 1500. Min. value is 68, max. value is 1500.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [IP]
# ARPENTRIES=Entries

Set the max. number of possible entries at the internal ARP/Routing table of the TCP/IP stack.

Valid Range: Between 32 and 512 .
Default value: 64.

Example: `ARPENTRIES=128`

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.21 | V1.21 | V1.21 | V1.06 | V1.00 |

Top of list
Index page

---

## [IP]
## ARPSMART=0/1

Set to 1 if the TCPIP stack should store all incoming ARP replies.
Set to 0 if the TCPIP stack should cache only incoming ARP replies, which are requested by itself.
Sc1x Default value: 1, ARPSMART enabled
Sc1x3/SC23 Default value: 0, ARPSMART disabled

Example: `ARPSMART=0`

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.21 | V1.21 | V1.21 | V1.06 | V1.00 |

Top of list
Index page

---

## [IPV6]
## Enable=0/1 Enable the IPv6 protocol

Enable generally the usage of IPv6 protocol. By default IPv6 is disabled.

### Related Topics

IPv6 stateless **auto configuration**
IPv6 address **initial value**
**IPV6CFG** - IPv6 configuration display command
Set IPv6 address **command**
Overview **IPv4/v6** - Dual layer stack

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

# [IPV6]
# ADDRESSx=IPv6 address of the Ethernet interface

Specify an IPv6 address for the Ethernet interface. It is possible to configure up to 5 fixed IPv6 addresses for the Ethernet interface.

## Comments

Example:

```
[IPV6]
ADDRESS0=3ffe:ffff:0:f101::1/64
ADDRESS1=4ffe:ffff:0:f101::1/64
```

The prefix length (e.g. the "/64" in above examples) must always be specified.

## Related Topics

Enable **IPv6** protocol
**IPv4** address initial value
IPv6 stateless **auto configuration**
**IPV6CFG** - IPv6 configuration display command
Set IPv6 address **command**
Overview **IPv4/v6** - Dual layer stack

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [IPV6]
# STATELESS_AUTOCONF=0/1 Stateless auto configuration

Disable/Enable stateless IPv6 auto configuration. By default stateless auto configuration is enabled.

## Comments

With stateless address auto configuration, a device interface is automatically assigned a link-local IPv6 address by the system. This link-local IPv6 address is generated with a fixed local address prefix (fe80::) to a token derived from the MAC address. (The address is verified to be unique.) This allows each IPv6 interface to have at least one source address that can be used by Neighbor Discovery. If an IPv6 router on the network advertises network prefixes in router advertisements, IPv6 derives IPv6 addresses based on the network interface identifier of the interface and on the network prefixes advertised.

There are 4 addresses left for further configurations by using the Neighbour discovery protocol.

## Related Topics

Enable **IPv6** protocol
IPv6 address **initial value**
**IPV6CFG** - IPv6 configuration display command
Set IPv6 address **command**
Overview **IPv4/v6** - Dual layer stack

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [IPSEC]
## ENABLE=0/1 Enable IP Security

Disable/Enable IP security. By default IP security is disabled. Note that the IP Security may also be started at run time with the **IPsec_Start()** API.

### Related Topics

**IKE** command
**IPsec_Start()** API
**IPsec_Set_Policy()** API
**NAT** enable

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## [IPSEC]
## POLICY_FILE=Restore IP Security Policies from file

This entry specifies a binary **file** that contains the IP Security policies and/or preshared keys to be installed either at startup (assuming IPSEC **ENABLE** set to 1) or at run time when **IPsec_Start()** API is called.

Example:

```
[IPSEC]
POLICY_FILE=ipsec.ips
```

## Comments

This file can be constructed with the **IPsec_Store_Policy** C-library function or with Beck's IP Security Policy Editor tool, BeckIPSec, available for use with a Windows PC. The file extension on this policy data file can be anything. The IPS extension is the convention used by the BeckIPSec tool.

## Related Topics

**IPsec_Store_Policy()** API
**tIPSEC_FILE_HEADER** Header format

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

# [IPSEC]
# PRESHARED_KEY=IKE Peer ID ; Key String

This entry specifies both the identity of an IKE (Internet Key Exchange) peer and an ASCII string to be installed for a preshared key used with that peer. A semicolon is used to delimit the two fields of this entry.

Example:

```
[IPSEC]
PRESHARED_KEY=192.168.30.145;MyPresharedKey#1
```

## Comments

Only one preshared key may be specified in this INI file. Additional keys may be added at runtime using the **IPsec_Add_Preshared_Key()** API or from a policy **file()**.

The peer identity can be a string specifying an IPv4 address, an IPv6 address, a fully qualified domain name or a user name. This string must match the manner that the peer uses to identify itself within the IKE protocol.

White space should be avoided immediately after the semicolon, otherwise this white space will be included as the first part of your key.

The total string entered after the "PRESHARED_KEY=" is limited to 260 characters. The string pair is truncated after 260 characters total.

## Related Topics

**IPsec_Add_Preshared_Key()** API
IPSEC **IKE_LOCALCERT** INI entry

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

# [IPSEC] IKE_PRIORITY=priority

This entry specifies the priority at which the Internet Key Exchange task, "_IKE", executes. Example:

```
[IPSEC]
IKE_PRIORITY=35
```

## Comments

Default priority is 50. This task can occupy 100% of the CPU for an expended period of time (seconds) during the key exchange processing. Consequently, this low default task priority is assigned to it.

## Related Topics

IPsec_Start() API
IPsec_Set_Policy() API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.10 | V1.00 |

Top of list
Index page

---

# [IPSEC] IKE_BUF_SIZE=Receiver buffer size

This entry specifies the size of the receiver buffer used by the Internet Key Exchange (IKE).

Example:

```
[IPSEC]
IKE_BUF_SIZE=4000
```

## Comments

The default IKE receiver buffer size is 2048 bytes. This may be too small for cases where a significant chain of certificates is to be received from the IKE peer. The sum of the certificate file sizes in the

anticipated CA chain can be used to judge the size of the required receiver buffer. Padding this sum by an additional 300 bytes should provide sufficient buffer space.

This entry will be limited internally to the range from 1024 bytes minimum up to 10000 bytes maximum.

Note that the **IKE** console command lists the peak IKE message size that has been received from the IKE peers. Appearance of the IKE **error** 100 indicates that you need to select a larger receiver buffer size with this INI file entry.

## Related Topics

**IKE** command

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

---

# [IPSEC]
# IKE_CACERTx=CA certificate file name

File name(s) for IKE CA certificates. IKE_CACERT0 must be the name of the IKE's root CA certificate. Up to four further CA certifications can be listed with tags IKE_CACERT1 up to IKE_CACERT4 to indicate a certification chain. Possible file formats are *.PEM and *.DER.

The system scans for these entries in CHIP.INI starting at IKE_CACERT0 and terminates the scan at the first non-existant entry in the sequence.

Key lengths longer than 4096 bits are not supported.

## Comments

Example:

[IPSEC]
IKE_CACERT0=ROOTCERT.DER

## Related Topics

IPSEC **IKE_LOCALCERT** INI entry
IPSEC **IKE_KEYFILE** INI entry

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list

# [IPSEC]
# IKE_CLICACERTx=File name of IKE peer's CA certificate

Up to five CA certificates for IKE peers may be specified with the 'x' in the above tag replaced with numbers '0' through '4'. The system scans for these entries in CHIP.INI starting at '0' and terminates the scan at the first non-existant entry.

Key lengths longer than 4096 bits are not supported.

## Comments

Example:

[IPSEC]
IKE_CLICACERT0=CACERT.DER

## Related Topics

IPSEC **IKE_CACERTx** INI entry

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

# [IPSEC]
# IKE_LOCALCERT=File name of our certificate

Defines the name of this system's local certificate. Possible file formats are *.PEM and *.DER.

## Comments

Example:

[IPSEC]
IKE_LOCALCERT=OurCERT.DER

If this INI file entry is present and the certificate is successfully loaded, then the IKE will use Public Key Infrastructure (PKI) authentication method in phase 1 of the key negotiation process unless:

1. The peer has initiated the IKE negotiation and has requested the use of **preshared key** for authentication.
2. The user has specified preshared key authorization method in the optional transform(s) set with the **IPsec_IKE_Phase1_Set()** API.

The **IPSEC_IKE_KEYFILE** and **IKE_CACERT0** files must also be specified for PKI operation.

## Related Topics

IPSEC **IKE_CACERTx** INI entries
IPSEC **IKE_KEYFILE** INI entry

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

**Top of list**
**Index page**

---

# [IPSEC]
# IKE_KEYFILE=File name of the IKE key file

Specifies the name of private and public key file. used for IKE. Possible file formats are *.PEM and *.DER.

Key lengths longer than 4096 bits are not supported.

## Comments

Example:

[IPSEC]
IKE_KEYFILE=PRIVKEY.DER

## Related Topics

IPSEC **IKE_LOCALCERT** INI entry

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

**Top of list**
**Index page**

---

# [IPSEC]
# IKE_PH1_AGGRESSIVE=IKE Phase 1 Aggressive mode

This entry specifies the Internet Key Exchange (IKE) should use aggressive mode for phase 1 SA negotiation.

Example:

```
[IPSEC]
```

```
IKE_PH1_AGGRESSIVE=1
```

## Comments

By default this option is deselected (value 0), which causes the phase 1 IKE negotiation to operate in the "main mode", also referred to as Identify Protection Exchange. (See RFC 2408 section 4.5.)

When this switch is set to 1 as shown in the example, then the phase 1 IKE will use the Aggressive Exchange which does not hide identifies. A few less messages must travel between computers in this case. (See RFC 2408 section 4.7.)

Note:

Microsoft operating systems have not supported this mode. You will likely get an IKE **error**code 7, "Invalid exchange type", when Aggressive mode is used with a Microsoft peer system.

## Related Topics

**IPsec_Set_Option()** API
**IPsec_IKE_Phase1_Set()** API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.20 | V1.20 |

**Top of list**
**Index page**

---

# [IPSEC]
# IKE_PH2_PFS=PFS mode for IKE phase 2

This entry specifies the PFS mode to be used by Internet Key Exchange (IKE) Quick Mode.

Example:

```
[IPSEC]
IKE_PH2_PFS=0
```

## Comments

Perfect Forward Secrecy (PFS) mode will be used during Quick Mode SA key negotiation when this switch is set to '1'. This is the default state of this switch.

PFS mode can be disabled by setting this switch value to zero.

Refer to RFC 2409 section 3.3 for a precise definition of PFS.

## Related Topics

**IPsec_Set_Option()** API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.20 | V1.20 |

Top of list
Index page

---

# [IPSEC]
# NAT=0/1 Enable IP Sec NAT-Transversal

Disable/Enable IP Security transversal of NAT devices (Network Address Translator). By default IP Security NAT-Transveral mode is enabled.

When this mode is enabled, the IKE will perform NAT detection and negotiate NAT-transversal mode with a peer as specified in RFC-3947 and use, when necessary, the UDP encapsulated ESP per RFC-3948 in order to provide IP security across the NAT. The UDP port 4500 is used for this purpose.

Normally it does no harm to leave this switch in its default enabled state. However there may be cases with some routers that interfere with the IKE operation as they try to "help" due to being IP Security aware, where it becomes necessary to disable the NAT-Transversal functionality in the IPC@CHIP® in order to get the IP Security to operate across a NAT.

Note that **AH protocol** (Authentication Header) is not supported when a NAT is present on the path between IPsec peers. The security **policy** must be set accordingly (ESP only!).

Note that the top level **IPSEC ENABLE** switch must be enabled for this switch to have affect.

## Related Topics

IPsec **NAT** operation
IPsec **ENABLE**
**NAT_TIMEOUT** keep-alive
**NAT_LINGER** keep-alive

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.16 | V1.01 |

Top of list
Index page

---

# [IPSEC]
# NAT_TIMEOUT=NAT Keep-Alive Packet Interval

Specify the interval in seconds at which NAT keep-alive UDP packets will be output when IP Security SA (Security Association) are established with the IPC@CHIP® lying behind a NAT device. (Ref: RFC 3948 section 4.)

```
[IPSEC]
```

```
NAT_TIMEOUT=30
```

## Comments

Default interval is 20 seconds. The range of legal entry values here is from 1 to 3600 seconds. Values outside this range will be changed to the nearest in range value.

## Related Topics

**NAT** enable
**NAT_LINGER** keep-alive

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.16 | V1.01 |

**Top of list**
**Index page**

---

# [IPSEC]
# NAT_LINGER=NAT Keep-Alive Linger Time

This entry specifies the number of minutes that the NAT keep-alive UDP packets will continue to be sent out after the related SA (Security Association) have been deleted. (Ref: RFC 3948 section 4.)

```
[IPSEC]
NAT_LINGER=6
```

## Comments

Default NAT keep alive linger time is 5 minutes. The range of legal entry values is from 1 to 720 minutes. Values outside this range will be changed to the nearest in range value.

The purpose of the NAT keep-alive UDP packets is to maintain the NAT address/port mappings. The linger time allows these mappings to be preserved as SA (Security Association) are re-negotiated after SA timeouts have occurred.

## Related Topics

**NAT** enable
**NAT_TIMEOUT** keep-alive

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.16 | V1.01 |

Top of list
Index page

# [UDPCFG]
# LEVEL=mask

Defines the supported functions of the configuration server.

LEVEL is a bit mask. Each bit represents one function. Setting the resp. bit to 1 enables the function, setting it to 0 disables the function.
The bit assignment is as follows:

- **Bit 0:** Detection of the IPC@CHIP® on the network
- **Bit 1:** IP configuration
- **Bit 2:** Password protection for IP configuration (see **here**)
- **Bit 3:** Reserved for future use
- **Bit 4:** Programming of flash (RTOS update)
- **Bit 5:** Password protection for programming of flash (see **here**)
- **Bits 6-14:** Reserved for future use
- **Bit 15:** This bit can be set to start the server without any function enabled. This could be useful if you want to set from an application at run-time.

The default value is 0x13. If the configuration level is set to 0x00, the configuration server task will not be started at all.

Example:
```
[UDPCFG]
LEVEL=0x03
```

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

## Comments

The password protection functions (bits 2 and 5) are only available on SC1x3 and SC23.

## Related Topics

Run-time adjustment of **configuration level**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.15 | V1.00 |

Top of list
Index page

# [UDPCFG]
# PRIO=prio

Defines the task priority of the configuration server (CFGS).

Legal values are in the range 2-127; default is 7.

Example:
```
[UDPCFG]
PRIO=30
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## [UDPCFG]<br>IPCFG_PASSWORD=password

Sets the password for IP configuration of the IPC@CHIP® via the configuration server. The password must consist of up to 16 printable non-whitespace characters.
Password protection must be enabled by setting the respective bit in the **configuration level**.

Examples:
```
IPCFG_PASSWORD=test
IPCFG_PASSWORD=abc123
IPCFG_PASSWORD=,zyx-.#321
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

---

## [UDPCFG]<br>FLASH_PASSWORD=password

Sets the password for flash programming of the IPC@CHIP® via the configuration server. The password must consist of up to 16 printable non-whitespace characters.
Password protection must be enabled by setting the respective bit in the **configuration level**.

Examples:
```
FLASH_PASSWORD=test
FLASH_PASSWORD=abc123
FLASH_PASSWORD=,zyx-.#321
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

---

## [DNS]
## NAME_SERVER1=IP address of the first name server

Adds a name server to the DNS resolver's name server list.
Normally the DNS resolver determines name server addresses automatically from network devices configured via DHCP or PPP. In addition you can specify two name servers in the CHIP.INI configuration file and via the function **DNS_SET_NAME_SERVER**. Name servers found via DHCP have the lowest priority. Next come name servers found via PPP, then the two configured in CHIP.INI. The name server added via function call has the highest priority.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

---

## [DNS]
## NAME_SERVER2=IP address of the second name server

See **NAME_SERVER1**.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

---

## [DNS]
## CACHE_SIZE=Number of entries of the DNS cache

Defines the number of entries in the DNS cache. More entries speed up DNS functions. But note that each entry consumes about 520 bytes of RAM memory. The default value is 30.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

---

## [PPPCLIENT]
## ENABLE=0/1

Disable/enable PPP client task

```
[PPPCLIENT]
ENABLE=1
```

By default, PPP client is enabled.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [PPPCLIENT]
## PRIO=prio

Defines the task priority of the PPP client task (PPPC).

Allowed value are 2-127. Default is 6.

Example:

```
[PPPCLIENT]
PRIO=8
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.15 | V1.00 |

Top of list
Index page

---

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

Disable/enable PPP server

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

By default, PPP server is disabled.

### Related Topics

PPP server **configuration** instructions

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [PPPSERVER]
## PRIO=prio

Defines the task priority of the PPP server task (PPPS).

Allowed value are 2-127. Default is 6.

Example:

```
[PPPSERVER]
PRIO=8
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.15 | V1.00 |

**Top of list**
**Index page**

---

## [PPPSERVER]
## IPV6_USE=0/1 Enable the IPv6 protocol

Enable the usage of IPv6 protocol at the PPP server and disable IPv4. By default IPv6 is disabled, the PPP server runs with IPv4.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

# [PPPSERVER]
# MODEMTRACE=0/1

Disable/enable the trace of the control communication between IPC@CHIP® and a connected modem. The modem strings (AT commands and answers) defined in chip.ini will be printed on STDOUT, if MODEM_TRACE=1.
This can be useful for testing the modem configuration and debugging the PPP dial procedures.

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

By default, tracing is disabled.

## Comments

If a command should be send from IPC@CHIP® to the modem, the MODEMTRACE output indicates only, that the command was successful placed at the send queue of the serial port. It doesn't indicate, that the command was physically send on the wire (e.g. if hardware flow control blocks the send queue). Received characters with an ASCII value smaller than 0x20 are printed as numbers.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Define your serial device for the PPP server.

**SC1x:**
Valid devices are COM or EXT.

**SC1x3:**
Valid devices are COM, EXT, SER2 or SER3.

**SC23:**
Valid devices are COM, SER2 or SER3.

## Comments

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

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

By default, no serial port is enabled.

## Related Topics

PPP server **configuration** instructions

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [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

PPP server **configuration** instructions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [PPPSERVER]
# GATEWAY=IP Address of gateway

Defines the IP address of the gateway if PPP Connection is established.

## Comments

Example: `GATEWAY=195.243.140.65`

If no gateway entry was found, the gateway will be untouched.

The TCP/IP stack of the IPC@CHIP® supports only one valid default gateway for all device interfaces: Ethernet, PPP Interface.

If you define a gateway in the PPPSERVER section of the `chip.ini`, it becomes the default gateway for all interfaces when a PPP link to the server is established. The default gateway must be the same IP Address as the remote peer.

It does not make sense to define a different IP, because the remote Peer is the only peer, which is reachable. If a different IP than the remote IP is defined, the remote IP will used for the gateway entry automatically.
During a PPP server connection the command **ipcfg** indicates this default gateway. After the PPP session, the old gateway (if any was defined) will be restored.

Also we provide some functions for modifying the 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

CHIP.INI entry **IP (Ethernet) GATEWAY**
PPP server **configuration** instructions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [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 **user name** / **password** pairs used to authenticate the PPP client. The client must use one of these pairs to get connected to the IPC@CHIP® PPP server.

## Related Topics

PPP server **configuration** instructions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [PPPSERVER] MTU=Bytes

Sets the Maximum Transfer Unit of the PPP server interface.

```
[PPPSERVER]
MTU=1300
```

By default, PPP server MTU is 1024. Min. value is 68.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

Top of list
Index page

---

# [PPPSERVER] IDLETIME=Seconds

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

```
[PPPSERVER]
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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [PPPSERVER] FLOWCTRL=0/1/2

Set flow control mode of the PPP server's serial device:

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

Example: Here XON/XOFF flow control is enabled

```
[PPPSERVER]
FLOWCTRL=1
```

By default, `FLOWCTRL=2` (RTS/CTS)

## Comments

**Caution:**
If you use the default DMA mode for the selected COM port, it is not recommended to choose XON/XOFF flow control mode (see **Fossil API Xon/XOff usage (Chap.3)**).

## Related Topics

PPP server **configuration** instructions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [PPPSERVER] 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [PPPSERVER]
# USERx=user

Define the user name for PPP server, using PAP or CHAP authentication

## Comments

You can define a USER0 and a USER1.
Default user is 'ppps', default password is 'ppps' for both USER0 and USER1.
You must specify both the user name and password.

If PAP authentication is used, neither user name nor password are case sensitive.
If CHAP authentication is used, user name and password specified here are converted to lower case.
The remote side must also specify the user name and password in lower case, if CHAP is used.

The entries are only valid if **AUTH** != 0 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [PPPSERVER]
# PASSWORDx=password

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

## Comments

You can define a PASSWORD0 for USER0 and a PASSWORD1 for USER1
Default user is 'ppps', default password is 'ppps'.

If PAP authentication is used, neither user name nor password are case sensitive.
If CHAP authentication is used, user name and password specified here are converted to lower case.
The remote side must also specify the user name and password in lower case, if CHAP is used.

The entries are only valid if **AUTH** != 0 is specified.

Maximum password size: 49 characters

## Related Topics

PPP server **configuration** instructions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [PPPSERVER] BAUD=BAUD Rate

Sets the BAUD rate of the PPP server serial port.

## Comments

The following example sets the PPP server serial 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [PPPSERVER] INITCMDx=modem command

Defines modem commands to initialize your modem connected to the IPC@CHIP®
at the start of the IPC@CHIP® PPP server and after a modem hang-up following a PPP session.

## Comments

You can define a maximum of 3 modem commands e.g. `INITCMD0=ATZ` .
The entries are only valid if **MODEM**=1 is specified.
The maximum length for each command string is 25 characters.

## Related Topics

PPP server **configuration** instructions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [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 3 modem answers e.g. `INITANSWER0=OK` .
The entries are only valid, if **MODEM**=1 is specified.
The maximum length for each answer string is 80 characters.

Default for all answer strings: OK

## Related Topics

PPP server **configuration** instructions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [PPPSERVER] MODEMCTRL=0/1/2

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [PPPSERVER]
## CTRLTIME=Seconds

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

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

By default, PPP server idle control time is 60 seconds.

If the PPP server doesn't receive regular PPP data during this interval, it executes the control commands . If execution of one of the control commands fails, the PPP server then closes the connection.

The CTRLTIME must be a smaller value than the **IDLETIME** of the PPP server.

### Related Topics

PPP server **configuration** instructions

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Defines modem command to control if modem is online or not at the start of the IPC@CHIP® PPP server

and after a modem hang-up following a PPP session.

### Comments

You can define a maximum of 3 modem commands e.g. `CTRLCMD0=+++` .
The maximum length for each control command string is 25 characters.

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

### Related Topics

PPP server **configuration** instructions

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [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 3 modem answers e.g. `INITANSWER0=OK` .
The entries are only valid, if **MODEMCTRL**=1 is specified.
The maximum length for each answer string is 80 characters.

### Related Topics

PPP server **configuration** instructions

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Defines 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Defines 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [PPPSERVER]

## CMDMODE=switch to modem command mode

Defines 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [PPPSERVER]

## HANGUPCMDx=modem command

Defines modem commands to hang-up the modem connected to the IPC@CHIP®.

### Comments

You can define a maximum of 3 modem hang-up commands e.g. `HANGUPCMD0=ATH` ,
which will be executed if the PPP connection is closed.
The maximum length for each command string is 25 characters.

### Related Topics

PPP server **configuration** instructions

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

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

### Comments

You can define a maximum of 3 modem answers e.g. `HANGUPANSWER0=OK` .
The maximum length for each answer message is 80 characters.
Default for all answer strings: OK

### Related Topics

PPP server **configuration** instructions

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Defines 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Defines 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [PPPSERVER] CONNECTMSGx=modem message

Defines 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` .
The maximum length for each connect message is 25 characters.

Defaults:
`CONNECTMSG0=RING`
`CONNECTMSG1=CONNECT`

## Related Topics

PPP server **configuration** instructions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Defines 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` .
The maximum length for each answer string is 80 characters.

Defaults: `CONNECTANSWER0=ATA`

## Related Topics

PPP server **configuration** instructions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [PPPSERVER]

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

Defines 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [RAMDRIVE]
## SIZE=size

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

**SC1x:**
Maximum size is 256 Kbyte

**SC1x3:**
Maximum size is 6144 Kbyte, if internal memory is used.
Maximum size is 8192 Kbyte, if external memory is used.
Default is to use internal memory.

**SC23:**
Maximum size is 6144 Kbyte.

### Comments

Default size: 0 (no RAM drive C:)

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.05 | V1.00 |

Top of list
Index page

---

# [RAMDRIVE]
# SEGMENT=segment

Set the start segment of the RAM drive C:.
If defined, the RAM drive memory is taken from external UCS# memory.
Default is to use internal memory.

Example:

```
[RAMDRIVE]
SEGMENT=0x8000
SIZE=512
```

## Comments

Segment must be greater or equal 0x8000.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | n/a |

Top of list
Index page

---

# [RAMDRIVE]
# WS=wait states

Set the number of wait states if RAM drive uses external memory.
Allowed values are 0-15
Default is to use 9 WS.

Example:

```
[RAMDRIVE]
WS=4
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | n/a |

Top of list
Index page

## [TIMER]
## 1C=ms

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

### Related Topics

BIOS Ints API **Set timer interrupt 0x1C's interval**

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [TIMER]
## AF=ms

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

### Related Topics

BIOS Ints API **Set timer interrupt 0xAF's interval**

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## [TIMER]
## RTI=kHz

**SC1x3/SC23:**

On SC1x3/SC23 systems, the Real-Time Interrupt (RTI) can be operated at rates above the normal 1000 Hz rate. This value is specified in kHz and can be set to any of the following legal rates:

[1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 25, 30, 32, 40, 48, 50]

## SC1x Comments

For SC1x targets, the RTI is fixed at a 1 kHz rate.

## SC1x3/SC23 Comments

The default rate of 1 kHz will be used for invalid entries not in the set of legal rates listed above. The RTI rate in effect is visible in the CX register returned by the **RTX_RTI_COUNT** API. The C-library function **RTX_Get_RTI_Rate** accesses this count.

If you will be using the C-library **RTX_GetTick_us** API, then for proper operation of this library function you are restricted to the following RTI settings:

[1, 2, 4, 5, 8, 10, 20, 25, 40, 50]

## Related Topics

**Using higher RTI rates**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

**Top of list**
**Index page**

---

# [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

**FTP enable/disable** command
BIOS Ints API Call **Suspend/Resume system servers**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**

Index page

---

# [FTP]
# PRIO=prio

Defines the task priority of the FTP server task (FTPS).

Allowed value are 2-127. Default is 41.

Example:

```
[FTP]
PRIO=80
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.15 | V1.00 |

Top of list
Index page

---

# [FTP]
# CMDPORT=port

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

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [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 reaches 20 seconds.
After a successful login the delay time will be set back to 400 milliseconds.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [FTP]
# TIMEOUT=sec

Defines the inactivity timeout for the FTP server in seconds.
The minimum value for the timeout is 20 seconds and maximum is 65535 seconds.

## Comments

Default FTP timeout is 300 seconds.
RFC 1123 states that the minimum idle timeout should be 5 minutes.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [FTP]
# USERx=user

Defines 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 their password.
Neither user name nor password are case sensitive.
Maximum name size: 19 characters

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

## Related Topics

FTP user **write** protection

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [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').
Neither user name nor password are case sensitive.
Maximum password size: 19 characters

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [FTP] ACCESSRIGHTx=Access rights for defined Users

This CHIP.INI entry allows you to deny write access to FTP USER0 or USER1.

0 - write and read access enabled
1 - write access denied, read access enabled

Example which disables FTP write access for USER0:

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

## Comments

You can only forbid write access if you have defined the respective user with the FTP **USERx** and

PASSWORDx entries.

By default write access is enabled for both FTP users.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [FTP]
# DRIVEx=Set user's FTP drive

Set user's FTP drive.

## Comments

Entries DRIVE0 or DRIVE1 can be made to specify a particular drive for use by FTP USER0 and USER1 respectively. The drive numbers are coded as follows:

0: Drive A (Default)
1: Drive B
2: Drive C
3: Drive D (SC1x3/SC23 only)

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

The following example defines the root drive for USER0 to be on B: drive.

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

## Related Topics

FTP user's root directory

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V1.10 | V1.00 |

Top of list
Index page

---

# [FTP]
# ROOTDIRx=Name of the user's FTP server root directory

Defines the name of user's FTP server root directory.

## Comments

The following example defines both root directory for USER1

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

The default FTP root directory is the drive root directory, "\". If the specified FTP directory doesn't exist, the FTP server closes the connection.

If ROOTDIRx is set you must also specify the FTP **DRIVEx** entry.

Maximum ROOTDIRx path string length: 64 characters

**Important notice:**

To avoid security leaks you should define one "normal" user with a directory below the "\" root directory. A user with the ROOTDIR setting "\" has access to every subdirectory on every drive. If the ROOTDIR is set to a subdirectory below the "\", the file access is restricted to the defined subdirectory and its subfolders on the specified drive **DRIVEx**.

## Related Topics

FTP user's **DRIVE**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [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.

## Related Topics

BIOS Ints API Call **Suspend/Resume system servers**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [WEB]
## PRIO=prio

Defines the task priority of the web server task (WEBS).

Allowed value are 2-127. Default is 41.

Example:

```
[WEB]
PRIO=80
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.15 | V1.00 |

Top of list
Index page

---

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

Defines 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [WEB]
## MAINPAGE_POST=Name of the main page (POST method)

Defines the name of Web server's main page when POST method is used. The Web server opens this

page if a browser request like `http://192.168.200.4/` is received. Typical names are "main.htm" or "index.htm". If this entry is not present, the **MAINPAGE** entry will specify the mainpage for both methods (GET and POST). The console command **webstat** shows the current main page.

## Related Topics

Set Web Server **Main** Page API Function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.20 | V1.20 |

Top of list
Index page

---

# [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 IPC@CHIP® RAMDISK by application programs.

The maximum string length is 32.

Example:

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

The **webstat** command shows the current temporary path.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Set Web server's disk drive.

0: Drive A
1: Drive B
2: Drive C
3: Drive D (SC1x3/SC23 only)

If the drive does not exist, the default drive A will be set.
The console command **webstat** shows the current Web server drive.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.10 | V1.00 |

Top of list
Index page

---

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

Defines 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 can read all your files.

### Related Topics

Set Web Server **Root** Directory API Function

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [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.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [WEB]
# WEBSERVERSTACK=Stack size

Sets the 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 such as `sprintf`, which require a lot of stack space, should increase the stack size to 6144 (6 KBytes).

The maximum value is 10240 bytes.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [WEB]
# HTTPPORT=port

Sets 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.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [WEB]

## POST_SIZE=size

Sets the size of the post method argument working buffer.
Default and minimum size is 768 byte, maximum is 8192 bytes. The web server allocates this memory for each of the active 4 connection handlers

Example:

```
[WEB]
POST_SIZE=1024
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | n/a | n/a |

Top of list
Index page

---

## [WEB]
## HEADER_SIZE=size in bytes

Sets the web server maximum header length in bytes. This applies to all lines in the HTTP header, the method line as well as all header fields.

Default header length: 1792 characters
Allowed value range: 256 - 5120

Example:

```
[WEB]
HEADER_SIZE=2000
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

---

## [WEB]
## HEADER_REJECT=0/1

Choose whether or not to reject incoming requests that contain headers longer than the internal buffer can store. If this is set to 0, and the received header is longer than the default or configured maximum header length, the corresponding header will be truncated. If this is set to 1 (default) the request will be rejected.

Example:

```
[WEB]
HEADER_REJECT=0
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

---

## [WEB]
## CONNECTIONS=Number of web server HTTP connections

Configure how many concurrent HTTP connections the web server can handle. Default on SC1x3/SC23 are 8 connections. Maximum connections are 30. This setting will heavily affect the TCP/IP memory pool usage amount.

Example:

```
[WEB]
CONNECTIONS=10
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

---

## [WEB]
## UPLOAD=0/1

Define if the Web server file upload should be activated.

### Comments

Use 0 to disable, 1 to enable.
By default, the Web server file upload is disabled.

### SC1x Comments

A special RTOS variant that supports the web file upload feature is required. In this special version the web file upload feature is always enabled.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.10 | V1.00 |

Top of list
Index page

---

## [WEB]
## USER0=User name for Web Server file upload

Defines a user name for transferring files to the Web server's root directory using the HTTP web file upload feature. The standard user name is 'WEB'.

The console command **webstat** shows the user name and password.

### Comments

Important notice: To avoid security leaks you should define a user name and password.

### SC1x Comments

A special RTOS variant that supports the web file upload feature is required.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V1.10 | V1.00 |

Top of list
Index page

---

## [WEB]
## PASSWORD0=Password for Web Server file upload

Defines the password used to transfer files to the Web server's root directory using the HTTP web file upload feature. The standard password is 'WEB'.

The console command **webstat** shows the Password and Username.

### Comments

Important notice: To avoid security leaks you should define a user name and password.

### SC1x Comments

A special RTOS variant that supports the web file upload feature is required.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V1.10 | V1.00 |

Top of list
Index page

---

# [WEB]
# SECURE=Activated the Web security feature for the Web Server

Defines whether the security feature for the Web Server is active or not.

0 = security feature deactivated (default)
1 = security feature activated

## Comments

The Web Server security feature allows up to two **paths** to be protected with **user name** and **password**. When this security feature is activated, users must then authenticate themselves to get Web access to these paths.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [WEB]
# SEC_URLx=Define a path for the security feature

Defines a specified URL for the Web security feature. The user can define SEC_URL0 and SEC_URL1.

All sub URLs of SEC_URLx are then protected by user name and password. The SEC_URL0 path is protected by **SEC_USER0** user name and **SEC_PASSWORD0** password, and SEC_URL1 by **SEC_USER1** user name and **SEC_PASSWORD1** password.

The maximum length for the paths is 63 characters.

## Comments

If the **security** feature is activated, the user should define a path, **user name** and **password**.

In the example below all sub URLs of "[IP]/sec" are protected (e.g. "192.168.200.4/sec/page.htm").

Example:

```
[WEB]
SEC_USER0=otto
```

```
SEC_PASSWORD0=web
SEC_URL0=/sec
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [WEB] SEC_USERx=Define a user name for the security feature

Defines a user name for the Web security feature.
The user can define SEC_USER0 and SEC_USER1.
The max length of the user name is 19 characters.

## Comments

If the **security** feature is activated, the user should define a **path**, user name and **password**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [WEB] SEC_PASSWORDx=Define a password for the security feature

Defines a password for the Web security feature.
The user can define SEC_PASSWORD0 and SEC_PASSWORD1.
The max length of the password is 19 characters.

## Comments

If the **security** feature is activated, the user should define a **path**, **user name** and password.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [WEB]
## SSL_ENABLE=Enable the SSL web server

Defines whether the SSL web server is used or not. IF SSL_ENABLE=1 the SSL web server task is started and running. IF SSL_ENABLE=2 the SSL web server task is started but suspended. In this case the user program must enable the server with **Suspend/Resume system servers**.

You have to provide a **CA certificate filename(s)**, a **local server certificate filename**, a **private and public key pair filename** and a optional **ephemeral key pair filename**.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.20 | V1.20 |

Top of list
Index page

---

## [WEB]
## SSL_PRIO=prio

Defines the task priority of the SSL web server task (SSLW).

Allowed value are 2-127. Default is 41.

Example:

```
[WEB]
SSL_PRIO=80
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

---

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

Set SSL Web server's disk drive.

0: Drive A
1: Drive B
2: Drive C
3: Drive D

If the drive does not exist, the default drive A will be set.
The console command **webstat** shows the current Web server drive.

If this entry is not present the normal Web server/s disk drive and root directory will be used.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.10 | V1.00 |

Top of list
Index page

---

## [WEB] SSL_ROOTDIR=Name of the SSL web server's root directory

Defines the name of SSL 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.

If the SSL_DRIVE entry is not present the normal Web server/s root directory will be used.

### Comments

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

---

## [WEB] SSL_HTTPPORT=port

Sets the port number of the SSL web server. Default SSL HTTP port: 443

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list

Index page

---

# [WEB]
# SSL_CONNECTIONS=Number of web server HTTPS connections

Configure how many concurrent HTTPS connections the web server can handle. Default on SC1x3/SC23 are 4 connections. Maximum connections are 30. This setting will heavily affect the TCP/IP memory pool usage amount.

Example:

```
[WEB]
SSL_CONNECTIONS=5
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

---

# [WEB]
# SSL_VERSION=Select the supported SSL version

Defines what SSL version the SSL web server should support.
1 = SSL 3.0
2 = TLS 1.0
3 = SSL 3.0 and TLS 1.0 (default)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [WEB]
# SSL_CACERTx=CA certificate filename

Defines the name of Web server's CA certificate. SSL_CACERT0 must be the name of the web server's root CA certificate. If needed SSL_CACERT1 is the next CA certificate in the certificate chain. SSL_CACERT2 is then the next in the chain and so on. Possible file formats are *.PEM and *.DER.

A maximume of five CA certificate files may be listed with tags SSL_CACERT0 up to SSL_CACERT4. The entry tag search is terminated at the first non-existant tag in this sequence.

## Comments

Example:

[WEB]
SSL_CACERT0=ROOTCERT.DER

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

---

## [WEB]
## SSL_CLICACERTx=File name of client's CA certificate

Defines the name of Web server client's CA certificate. If this entry is present, the web server will request a client authentication in his ServerHello message. The server will list all CAs added by these entries in his message.

A maximume of five CA certificate files may be listed with tags SSL_CLICACERT0 up to SSL_CLICACERT4. The entry tag search is terminated at the first non-existant tag in this sequence.

### Comments

Example:

[WEB]
SSL_CLICACERT0=CLICA.DER

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

---

## [WEB]
## SSL_SERVERCERT=File name of the server certificate

Defines the name of Web server's own local certificate. Possible file formats are *.PEM and *.DER.

### Comments

Example:

[WEB]
SSL_SERVERCERT=SRVCERT.DER

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [WEB]
# SSL_KEYFILE=Filename of the server key file

Defines the name of Web server's private and public key file. Possible file formats are *.PEM and *.DER.

Key lengths longer than 4096 bits are not supported.

## Comments

Example:

[WEB]
SSL_KEYFILE=PRIVKEY.DER

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [WEB]
# SSL_EPHFILE=Filename of the ephemeral key file

Defines the name of Web server's ephemeral private and public key file. Possible file formats are *.PEM and *.DER. When exportable cipher suite is used and the server's certificate is RSA with key length more than 512 bits, we can not use that RSA key pair to exchange keys, because the exportable cipher suite is only allowed to have a maximum key size of 512 bits. Instead we have to use ephemeral RSA keys. To generate the RSA key pair in real-time would have cost an excessive amount of CPU time, that's why this is not supported. In these cases the user must add ephemeral RSA keys, with a maximum size of 512 bits, using this entry.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [SSH]
# ENABLE=Enable the SSH server

Defines whether the SSH server is used or not.  IF ENABLE=1 the SSH server is started. You have to provide at least one key filename. A **RSA Key filename**, and/or **DSA Key filename**.

To enable STDIO on SSH sessions, you have to set **STDIN** and **STDOUT** to SSH.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

# [SSH]
# SSHPORT=port

Sets the port number of the SSH server.

Default SSH port: 22

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

# [SSH]
# PRIO=prio

Defines the task priority of the SSH server tasks (SSHx) during the session.

Allowed value are 2-127. Default is 11.

Example:

```
[SSH]
PRIO=11
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

# [SSH]
# PRIO_KEX=prio

Defines the task priority of the SSH server tasks (SSHx) during the key exchange.
The key exchange will occupy 100% of the CPU performance for a long time period. Thus the task will have a different task priority during the key exchange.

Allowed value are 2-127. Default is 50.

Example:

```
[SSH]
PRIO_KEX=60
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

# [SSH]
# RSA_KEYFILE=Filename of the SSH server RSA key file

Defines the name of SSH server's private key file, used to authenticate the server to the client. The file must be in the 'OpenSSH' format.

Key lengths longer than 4096 bits are not supported.

## Comments

Example:

```
[SSH]
RSA_KEYFILE=rsa_key
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list

Index page

---

# [SSH]
# DSA_KEYFILE=Filename of the SSH server DSA key file

Defines the name of SSH server's private key file, used to authenticate the server to the client. The file must be in the 'OpenSSH' format.

Key lengths longer than 4096 bits are not supported.

## Comments

Example:

[SSH]
DSA_KEYFILE=dsa_key

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

# [SSH]
# USERx=user

Defines a user name for SSH.

You can define a USER0 and a USER1.
There's no default user name and password.
User name and password are case sensitive.

Maximum user name size: 19 characters

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

# [SSH]
# PASSWORDx=password

Define the password for a SSH user.

You can define a PASSWORD0 for USER0 and a PASSWORD1 for USER1.
There's no default user name and password.
User name and password are case sensitive.

Maximum password size: 19 characters

## Comments

If no password is specified for a user, public key authentication will be the only allowed user authentication method. In this case the section **USERx_KEYFILE** must be provided.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

# [SSH]
# USERx_KEYFILE=File name with user public keys

Defines a filename for SSH USER0 or USER1, that contains one or multiple public keys for user authentication. This is an alternate method for user authentication, beside the password method. It is more secure than the password method.

If the SSH server should allow password and public key authentication, both entries (PASSWORDx and USERx_KEYFILE) must be provided.

The file must be in 'OpenSSH authorized_keys' format. Every key must be listed in one line. Within one key no line breaks are allowed.

Key lengths longer than 4096 bits are not supported.

## Comments

Example:

```
[SSH]
USER0=myself
USER0_KEYFILE=user0_keys
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## [SSH]
## LOCAL_FORWARD=0/1

Enables/Disables the SSH Local TCP Port Forwarding. By default the Local Forwarding option is disabled.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## [SSH]
## REMOTE_FORWARD=0/1

Enables/Disables the SSH Remote TCP Port Forwarding. By default the Remote Forwarding option is disabled.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## [SSH]
## FORWARD_ALL=0/1

Allows/Disallows TCP connections to remote forwarded ports from any host. By default only connections from the Local Loopback device are allowed (forwarded over SSH).

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## [SSH]
## REKEY_DATA=Data size before rekeying

Sets the transfered (sent + received) data size, before rekeying is requested. A value of 0 will disable the rekeying by reason of data size.
The default data size is 1073741824 Bytes (1 GB).

### Comments

Remember that the SSH client can also request rekeying. Most clients also allow to configure this setting.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

Top of list
Index page

---

## [SSH]
## REKEY_TIME=Time before rekeying

Sets the time in milliseconds, before rekeying is requested. A value of 0 will disable the rekeying by reason of timeout.
The default time is 28800000 ms (3600000*8 = 8 hours).

### Comments

Remember that the SSH client can also request rekeying. Most clients also allow to configure this setting.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

Top of list
Index page

---

## [TFTP]
## TFTPPORT=port

Sets the port number of the TFTP server.
Default TFTP port: 69

Example:

```
[TFTP]
TFTPPORT=4000
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| on request | on request | on request | V0.90 | V1.00 |

Top of list
Index page

---

# [TFTP]
# PRIO=prio

Defines the task priority of the TFTP server task (TFTD).

Allowed value are 2-127. Default is 41.

Example:

```
[TFTP]
PRIO=80
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| on request | on request | on request | V1.15 | V1.00 |

Top of list
Index page

---

# [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.

## Related Topics

BIOS Ints API Call **Suspend/Resume system servers**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## [TELNET]
## PRIO=prio

Defines the task priority of the Telnet server task (TELN).

Allowed value are 2-127. Default is 11.

Example:

```
[TELNET]
PRIO=10
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.15 | V1.00 |

Top of list
Index page

---

## [TELNET]
## TELNETPORT=port

Sets the port number of the Telnet server.
Default Telnet port: 23

Example:

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [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)

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [TELNET]
# LOGINDELAY=0/1

Define if the delayed login of the Telnet server should be 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.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [TELNET]
# LOGINRETRIES=number of login retries

Defines the number of login retries until Telnet session will be closed.

Example:

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

## Comments

The default value is 5.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list

Index page

---

# [TELNET]
# USERx=user

Defines a 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 their password.
Neither user name nor password are case sensitive.

Maximum user name size: 19 characters

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [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'.
Neither user name nor password are case sensitive.

Maximum password size: 19 characters

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [DEVICE]

## FILESHARING=0/1

Disable/Enable the file sharing. See also BIOS Interrupt service **0x37**.

### Comments

0=disable (default), 1=enable

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [DEVICE] POWERSAVE=1/2/4

Set the used power save mode. See also BIOS Interrupt service **0x18**.

### Comments

1=run with full speed (default), 2=run with 1/2 clock, 4=run with 1/4 clock

### SC1x3/SC23 Comments

If POWERSAVE=4, the internal Ethernet controller runs fixed in 10MBit, Half duplex mode.
This entry also influences the USB API. If POWERSAVE=2 the USB API cannot be used at all. The SC1x3/SC23 will crash if you do. If POWERSAVE=4 the USB API can only be used in Device mode, because Host mode needs better performance.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | V1.00 | V1.00 | V1.06 | V1.00 |

Top of list
Index page

---

## [DEVICE] PFI_ENABLE=0/1

Enable PIO9 as PFI pin, so the system recognizes the power fail interrupt. The PFI pin is checked during boot. The RTOS boot process will be suspended as long as the PFI signal is high. This assures that the system only starts with a valid voltage.
Default is not to use the PFI pin.

## Comments

WARNING: Don't enable this pin, if you don't have a power fail circuit connected to the PFI pin, else the chip will not boot and the RTOS will not come up anymore, because the PIO9 pin, if unconnected, is normally high (input with pullup).

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [TRACE]
# FLASHWRITE=0/1

Trace the activity of flash writes for debug purposes.

## Comments

Set to 1 to enable the debug output. On STDOUT a message is printed at every call to the low-level physical flash write routine.

By default the debug output is disabled.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [TRACE]
# INTNOTSUPP=0/1

Trace the activity of calls to not supported interrupts or functions.

## Comments

Set to 0 to disable the debug output. On STDOUT a message is printed at every call to a not supported interrupt vector or not supported function.

By default the debug output is enabled.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [SERIAL]
# EXT_DMA=0/1

Disable/enable DMA receive mode on the serial port 0 (EXT). If DMA receive mode is disabled, the serial port 0 (EXT) 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 serial port 0 (EXT) if the DMA channel is needed by an external device. See all possible **DMA configurations**. In the IRQ receive mode, you may loose 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 serial port 0 (EXT).

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

By default, DMA receive mode is enabled.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

---

# [SERIAL]
# COM_DMA=0/1

Disable/enable DMA receive mode on the serial port 1 (COM). If DMA receive mode is disabled, the serial port 1 (COM) 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 serial port 1 (COM) if the DMA channel is needed by an external device. See all possible **DMA configurations**. In the IRQ receive mode, you may loose 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 serial port 1.

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

By default, DMA receive mode is enabled.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [SERIAL]
# SER2_DMA=0/1

Disable/enable DMA receive mode on the serial port 2. If DMA receive mode is disabled, the serial port 2 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 serial port 2 if the DMA channel is needed by an external device. See all possible **DMA configurations**. In the IRQ receive mode, you may loose 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 serial port 2.

```
[SERIAL]
SER2_DMA=0
```

By default, DMA receive mode is enabled.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list

# [SERIAL]
# SER3_DMA=0/1

Disable/enable DMA receive mode on the serial port 3. If DMA receive mode is disabled, the serial port 3 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 serial port 3 if the DMA channel is needed by an external device. See all possible **DMA configurations**. In the IRQ receive mode, you may loose 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 serial port 3.

```
[SERIAL]
SER3_DMA=0
```

By default, DMA receive mode is enabled.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

# [SERIAL]
# SEND_DMA=0/1/2

Selects the DMA send mode for a serial port.

**SC12/SC1x3:**
Use 0 to enable the DMA send mode for the serial port 0 (EXT).
Use 1 to enable the DMA send mode for the serial port 1 (COM).

**SC23:**
Use 1 to enable the DMA send mode for the serial port 1 (COM).

**SC11/SC13:**
Use 0 to enable the DMA send mode for the serial port 0 (EXT).
Use 1 to enable the DMA send mode for the serial port 1 (COM).
Use 2 to enable the DMA send mode for both ports.

Example which enables the DMA send mode on the serial port 1 (COM).

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

By default, DMA send is disabled.

## SC12/SC1x3 Comments

**Important:**
By default both useable DMA channels for the serial ports EXT and COM are used for receiving characters. 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) automatically switches over to using the serial port's receiver interrupt.

**Possible DMA configurations:**

| COM_DMA | EXT_DMA | SEND_DMA | | DMA1 | DMA0 |
|---|---|---|---|---|---|
| 0 | 0 | not present | | DRQ1 | DRQ0 |
| 0 | 1* | not present | | DRQ1 | RECV EXT |
| 1* | 0 | not present | | RECV COM | DRQ0 |
| 1* | 1* | not present | | RECV COM | RECV EXT |
| don't care | 0 | 0 | | SEND EXT | DRQ0 |
| don't care | 1* | 0 | | SEND EXT | RECV EXT |
| 0 | don't care | 1 | | DRQ1 | SEND COM |
| 1* | don't care | 1 | | RECV COM | SEND COM |

1* = Value 1 is default so the entry must not be present

## SC23 Comments

**Important:**
By default both the DMA channel for the serial port COM is used for receiving characters.

**Possible DMA configurations:**

| COM_DMA | SEND_DMA | | DMA1 | DMA0 |
|---|---|---|---|---|
| 0 | not present | | not used | not used |
| 1* | not present | | RECV COM | not used |
| 0 | 1 | | not used | SEND COM |
| 1* | 1 | | RECV COM | SEND COM |

1* = Value 1 is default so the entry must not be present

## SC11/SC13 Comments

**Important:**
The IPC@CHIP®'s SC11, SC13 have four DMA channels, which are available for the serial ports EXT and COM. By default two DMA channels are used for receiving characters from the COM and EXT ports.

**Possible DMA configurations:**

| COM_DMA | EXT_DMA | SEND_DMA | | DMA3 | DMA2 | DMA1 | DMA0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | not present | | not used | not used | DRQ1 | DRQ0 |
| 0 | 1* | not present | | not used | RECV EXT | DRQ1 | DRQ0 |
| 1* | 0 | not present | | RECV COM | not used | DRQ1 | DRQ0 |
| 1* | 1* | not present | | RECV COM | RECV EXT | DRQ1 | DRQ0 |
| 0 | 0 | 0 | | SEND EXT | not used | DRQ1 | DRQ0 |
| 0 | 0 | 1 | | not used | SEND COM | DRQ1 | DRQ0 |
| 0 | 1* | 0 | | SEND EXT | RECV EXT | DRQ1 | DRQ0 |
| 0 | 1* | 1 | | not used | RECV EXT | DRQ1 | SEND COM |
| 1* | 0 | 0 | | RECV COM | not used | SEND EXT | DRQ0 |
| 1* | 0 | 1 | | RECV COM | SEND COM | DRQ1 | DRQ0 |
| 1* | 1* | 0 | | RECV COM | RECV EXT | SEND EXT | DRQ0 |
| 1* | 1* | 1 | | RECV COM | RECV EXT | DRQ1 | SEND COM |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 2 | | SEND EXT | SEND COM | DRQ1 | DRQ0 |
| 0 | 1* | 2 | | SEND EXT | RECV EXT | DRQ1 | SEND COM |
| 1* | 0 | 2 | | RECV COM | SEND COM | SEND EXT | DRQ0 |
| 1* | 1* | 2 | | RECV COM | RECV EXT | SEND EXT | SEND COM |

1* = Value 1 is default so the entry must not be present

## SC1x Comments

Note that **RS485** is not available with serial send DMA.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [SERIAL] SEND_DMA2=0/1

Selects the DMA send mode for a serial port.

## Comments

Use 0 to enable the DMA send mode for the serial port 2.
Use 1 to enable the DMA send mode for the serial port 3.

Example which enables the DMA send mode on the serial port 3.

```
[SERIAL]
SEND_DMA2=1
```

By default, DMA send is disabled.

## SC1x3/SC23 Comments

**Important:**
By default the two DMA channels for port 2 and 3 are used for receiving characters from the ports 2 and 3. If you want to use the DMA send mode for one serial port (e.g. the port 2), the second port (e.g. the port 3) automatically switches over to using the serial port's receiver interrupt.

**Possible DMA configurations:**

| SER3_DMA | SER2_DMA | SEND_DMA2 | | DMA3 | DMA2 |
|---|---|---|---|---|---|
| 0 | 0 | not present | | DRQ3 | DRQ2 |
| 0 | 1* | not present | | DRQ3 | RECV port2 |
| 1* | 0 | not present | | RECV port3 | DRQ2 |
| 1* | 1* | not present | | RECV port3 | RECV port2 |
| don't care | 0 | 0 | | SEND port2 | DRQ2 |
| don't care | 1* | 0 | | SEND port2 | RECV port2 |

| | | | | | |
|---|---|---|---|---|---|
| 0 | don't care | 1 | | DRQ3 | SEND port3 |
| 1* | don't care | 1 | | RECV port3 | SEND port3 |

1* = Value 1 is default so the entry must not be present

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## [SERIAL]
## EXT_RECVQUEUE=size

Sets 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.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

---

## [SERIAL]
## EXT_SENDQUEUE=size

Sets 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.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

---

## [SERIAL]
## COM_RECVQUEUE=size

Sets 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 be set to 4096.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [SERIAL]
## COM_SENDQUEUE=size

Sets 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 be set to 4096.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [SERIAL]
# SER2_RECVQUEUE=size

Sets the receive queue size of the serial port2. Minimum size is 1024.
Maximum size is 10240 byte.

## Comments

Example:
```
[SERIAL]
SER2_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 be set to 4096.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [SERIAL]
# SER2_SENDQUEUE=size

Sets the send queue size of the serial port 2. Minimum size is 1024.
Maximum size is 10240 byte.

## Comments

Example:
```
[SERIAL]
SER2_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 be set to 4096.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [SERIAL]
# SER3_RECVQUEUE=size

Sets the receive queue size of the serial port 3. Minimum size is 1024.
Maximum size is 10240 byte.

## Comments

Example:
```
[SERIAL]
SER3_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 be set to 4096.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# [SERIAL]
# SER3_SENDQUEUE=size

Sets the send queue size of the serial port 3. Minimum size is 1024.
Maximum size is 10240 byte.

## Comments

Example:
```
[SERIAL]
SER3_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 be set to 4096.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list

Index page

---

# [SERIAL]
# COM_BAUD=BAUD Rate

Sets 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).

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# [SERIAL]
# EXT_BAUD=BAUD Rate

Sets 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).

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

---

# [DOSLOADER]
# MEMGAP=Paragraphs

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

## Comments

Some programs compiled with Borland C 5.02, Beck Paradigm Edition (other compilers??) try to increase their program memory block at runtime. This can occur, for example, when opening a file with Borland/Paradigm C-library function `fopen`, where some additional memory is required. The C-library `fopen` function calls int 21h **0x4A**, which is not directly visible to the application programmer. This memory resize call fails if another program is loaded after the previous one, because now there is no memory space left for increasing the memory size of the previously executed program. The program then returns from `fopen` with an error. In this case, the global program variable errno is set to value 8 (not enough memory).

To prevent this error, the @CHIP-RTOS allows a memory gap of a defined size between loaded programs. This memory gap size is specified as a number of paragraphs(where 1 paragraph equals 16 bytes for SC1x, and 256 bytes for SC1x3/SC23 systems).
This strategy can fail when programs are terminated and restart again.

Example:

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

By default, MEMGAP is set to 0. The maximum value is 2048 paragraphs, where any value larger than this is truncated 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. Only if a C-library function call sets errno to 8, should this value be defined. We recommend in that case a value of 128 paragraphs (2048 Bytes). The described problem was noticed when the Borland C-library function `fopen` was used. The same can happen with usage of C-library function `malloc` using memory model Large. The `malloc` returns a NULL pointer in this case.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [BATCH]
# BATCHMODE=0/1

Sets the **batch file** execution mode of 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 @CHIP-RTOS 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 @CHIP-RTOS Interrupt **0xA0 Service 0x15**, which immediately enables the further batch file sequencing.

## Related Topics

**BATCHMODE** command
Run-time **batch mode** selection API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# [BATCH]
# EXECTIMEOUT=0/1

Disable (=0) / enable (=1) the **batch file** program execution delay time limit for BATCHMODE=1.
`EXECTIMEOUT=0` :
The successor program in a batch file waits forever if the predecessor program neither finishes nor calls **0xA0 Service 0x15**.

`EXECTIMEOUT=1` : (Default mode)
The maximum delay time for execution of the next listed program in a batch file is 15 seconds.

## Comments

This Boolean control applies only to BATCHMODE=1.

Example:

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

## Related Topics

**BATCHMODE** Configuration

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## [USB]
## MAX_POWER_SUPPLY=Maximum power supplied via USB

Use this entry to tell the RTOS USB-driver how much power can be supplied to a bus-powered device in Host mode. The value is given in mA. The default value is 500. This means that if you don't specify a value here, the RTOS USB-driver assumes that your hardware is able to supply up to 500 mA to a USB device.

Example:

```
[USB]
MAX_POWER_SUPPLY=100
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## [USB]
## BULK_PER_FRAME=Maximum number of BULK transfers

This entry defines the maximum number of transfers that will be performed for each BULK endpoint per frame. A higher value increases the bandwidth of each BULK endpoint. If you for example set this entry to 8, a maximum of 500 kb/sec can be transfered on an endpoint with a maximum packet size of 64 (8 * 64 bytes per millisecond).
However a higher value also increases the number and duration of USB interrupts. This may lead to decreasing performance.

Valid values are between 1 and 12. The default value is 6.

Example:

```
[USB]
BULK_PER_FRAME=8
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **n/a** | **n/a** | **n/a** | **V1.20** | **V1.20** |

**Top of list**
**Index page**

End of document




# @CHIP-RTOS C Library V2.15 - Index

***CLIB Documentation Index:***

**BIOS API**
**CAN (Controller Area Network) API**
**CGI (Common Gateway Interface) API**
**External Disk API**
**Fossil API for Serial Ports**
**Helper Functions**
**Hardware (PFE, HAL) API**
**I2C Bus Interface API**
**Packet Driver API**
**RTOS API**
**SPI API**
**TCP/IP API**
**USB (Universal Serial Bus) API**

***RTOS Documentation Index:***

**@CHIP-RTOS Main Index**

---

## Memory Models Available:

The API library comes in memory models:

- CLIBxxxS.LIB - Small model
- CLIBxxxM.LIB - Medium model
- CLIBxxxC.LIB - Compact model
- CLIBxxxL.LIB - Large model
- CLIBxxxH.LIB - Huge model

This is the compiler memory model. It is not associated with the different RTOS version (TINY, MEDIUM, LARGE, ...)

---

## Library applies to all @CHIP-RTOS versions lower than or equal to:

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.22 | V1.22 |

---

End of document

---




# @CHIP-RTOS - Software Interrupt Index

***Software Interrupt Documentation Index:***
**BIOS API: Interrupts for several PC services**
**CAN API: CAN Bus Driver**
**CGI/SSI API: Web Server CGI/SSI Interface and File upload**
**DOS API: Interrupt 0x21 functions**
**Ethernet: Packet Driver Interface**
**External Disk: External Drive Interface Definition**
**FOSSIL API: Interface to the serial ports**
**Hardware API: Including PFE and HAL**
**I2C / SPI API: Interface definition for the I2C Bus and SPI Bus Interface**
**RTOS API: Interface definition for RTOS interface**
**TCP/IP API: Interface definition for the TCP/IP sockets**
**TCP/IP Device Driver: User specific device driver/linklayer**
**USB API: Universal Serial Bus Driver**

***RTOS Documentation Index:***
**@CHIP-RTOS Main Index**

---

## Applies to all @CHIP-RTOS versions lower than or equal to:

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.22 | V1.22 |

---

End of document

---




# @CHIP-RTOS - PPP Interface



## Introduction

Here is a short description of how to configure the IPC@CHIP® PPP server.
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**
- **IPv6 configuration of the PPP Server**
- **PPP Server API**
- **PPP Client API**
- **Available Examples**


## About PPP

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 IPC@CHIP® 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 a considered the client. The machine to which you connect (e.g. the IPC@CHIP®) is considered the server.

PPP on 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 serial port for the PPP server to use with the **COMPORT**directive.

   **Important** :
   The COM and EXT port of the IPC@CHIP®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 serial port.

5. Set the baud rate of PPP server's serial 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.

   There are three different possibilities for configuring PPP server IP addresses on the IPC@CHIP®:

   a) If valid addresses for **IP Address**and **REMOTE IP Address**are declared in chip.ini, the PPP server wants to use this configured IP for its own and wants to configure the remote peer with the defined remote address.

   b) If only **IP Address**is declared in chip.ini and **REMOTE IP Address**is set to 0.0.0.0, the PPP server wants to use this configured IP address and expects the client to use their own address.

   c) 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:
   The TCP/IP stack of the @CHIP-RTOS supports different network interfaces, Ethernet and PPP, so the IPC@CHIP®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 @CHIP-RTOS 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 IPC@CHIP® @CHIP-RTOS 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 IPC@CHIP® 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 IPC@CHIP® boot process.

`. Switch the modem into the command mode (**CMDMODE** and **HANGUPDELAY**)

Example:

```
CMDMODE=+++
HANGUPDELAY=2
```

These are the default values.

a. 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 IPC@CHIP® 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 or **MODEMCTRL**=2

The default is 0 (disabled).
Also you have to configure a control time interval (in seconds). Two different modes are provided:

MODEMCTRL=1:
After each time interval during which the PPP server receive no data, the server executes the configured modem commands.

MODEMCTRL=2:
After each time interval the server executes the configured modem commands.
In both cases the server closes the connection, if one of the expected answers timed out.
In this mode the configured modem commands are not printed via stdout at execution time, even if **MODEMTRACE** is enabled.

Example:

```
MODEMCTRL=1
```

a. Define a **CTRLTIME**.
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 IPC@CHIP® 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**

---

## IPv6 configuration of the PPP Server

PPP supports IPv6 with an additional Network control protocol (IPV6CP).
IPv6 address configuration is done by a auto negotiation process. At connection time the peers negotiate an Interface-ID for the PPP device and auto configure their link-local scope addresses. The address configuration and other configuration data are available by calling **Get PPP server configuration** .

**Top of list**
**Index page**

---

## PPP Server API

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
6. Interrupt 0xAC, **Service 0x55**: Set the PPP negotiate options

**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
5. Interrupt 0xAC, **Service 0x44**: Get PPP primary and secondary DNS IP addresses
6. Interrupt 0xAC, **Service 0x45**: Set the PPP negotiate options

**Top of list**
**Index page**

---

## Available Examples

Source code examples can be found at **www.beck-ipc.com**.

**Top of list**
**Index page**

---

End of document

---




# @CHIP-RTOS - Web server



## Introduction

Here is a short description of the IPC@CHIP® Web server.

The IPC@CHIP® web server provides information, e.g. HTML pages, over the Hypertext Transfer protocol (HTTP). The server can be accessed with any normal web browser.

---

**Features:**

- Static HTML pages, no special tool for generation required, just upload to file system
- Dynamic HTML (**CGI**) pages, implemented as callback functions
- Configurable TCP port number (default 80)
- Web file upload support available through special RTOS variant

---

**Features: (SC1x3/SC23 only)**

- SSL web server (HTTPS) functionality
- Configurable SSL TCP port number (default 443)
- Server Side Include (SSI) pages, implemented as callback functions
- Support for Cookies
- Web file upload support included in standard RTOS version

The web server can be configured with the corresponding WEB **CHIP.INI entries**. The server can be suspended/resumed with the **BIOS_Server_On_Off()** API

---

End of document

---




# @CHIP-RTOS - FTP server



## Introduction

Here is a short description of the IPC@CHIP® FTP server.

The FTP (File Transfer Protocol) server allows file transfers to or from a remote FTP client over the network. Once connected, the client can do a number of file manipulation operations such as uploading files to the server, download files from the server, rename or delete files on the server and so on.

---

**Features:**

- Supports only one active connection (only client at one time)
- Supports active and passive mode
- Two available user accounts with configurable access rights
- Configurable TCP port number (default 21)
- Configurable idle timeout

Default users are: 'anonymous' (no password) and 'ftp' (password is 'ftp'). The FTP server can be configured with the corresponding FTP **CHIP.INI entries**. It can also be enabled/disabled with the **FTP** shell command. This shell command can be executed from within an application using the **BIOS_Execute()** API. The server can be suspended/resumed with the **BIOS_Server_On_Off()** API

---

End of document




# @CHIP-RTOS - TFTP server



## Introduction

Here is a short description of the IPC@CHIP® TFTP server.

The TFTP server allows sending and receiving of files to a remote TFTP client. (Thus the term *Trivial File Transfer Protocol* , TFTP.)

For security reasons, TFTP file transfers are disabled by default. You can enable/disable TFTP with the TFTP 0/1 shell command **TFTP**. This shell command can be executed from within an application using the **BIOS_Execute()** API.

TFTP can be used as a simple alternative to FTP. TFTP does not provide extended file system features like listing directories or deleting files. If the user does not need these extended features of the FTP server, a Sc1x @CHIP-RTOS variant with TFTP instead of the FTP may be useful.

By default the server listens at the standard TFTP port 69 for incoming TFTP client requests. The TFTP port can be **configured** in chip.ini.

The server is able to serve only one client at one time. Only the binary (octet) transfer mode is supported. The size of TFTP data packets must be 512 bytes.

## SC1x Comments

The TFTP server is not a part of the six official Sc1x @CHIP-RTOS variants. You must directly order a special SC1x @CHIP-RTOS version including this feature.

## SC1x3/SC23 Comments

At SC1x3/SC23 @CHIP-RTOS versions, the TFTP server is always available.

End of document




# @CHIP-RTOS - Telnet server



## Introduction

Here is a short description of the IPC@CHIP® Telnet server.

The Telnet protocol is used for interactive remote access over the network. Once the connection with a Telnet client is established, the user can log in with his account information and execute operating system commands remotely on the IPC@CHIP®, such as *dir* or *cd* .

**Features:**

- Supports only one active connection (only one client at one time)
- Two available user accounts
- Configurable TCP port number (default 23)
- Configurable timeout

Default user are 'tel' (password is 'tel') for both accounts. The Telnet server can be configured with the corresponding Telnet **CHIP.INI entries**. The server can be suspended/resumed with the **BIOS_Server_On_Off()** API

End of document

# @CHIP-RTOS - Secure Shell

IPC@CHIP® Documentation **Index**

## SSH Server - Secure Shell

This documentation describes the SSH Server (Secure Shell) of the @CHIP-RTOS. The SSH Server is only available on the IPC@CHIP®s SC1x3/SC23. The goal of SSH is to replace the earlier Telnet protocol, which did not provide strong authentication or guarantee confidentiality.

---

**Features:**

- SSH-2 protocol
- **OpenSSH** compatible key format
- Password and public key user authentication
- Local and Remote TCP Forwarding support
- Cryptographic algorithms: Diffie-Hellman, RSA, DSA, AES, ARCFOUR, 3DES, MD5, SHA-1
- Number of supported sessions (clients): 5
- Number of supported channels per session: 10

---

**Configuration:**

By default the SSH server is disabled.
The user must enable and configure the server with some **SSH CHIP.INI entries**.
For key generation we suggest to use **PuTTY**.

Create the server key for server authentication:

- Open 'puttygen.exe'.
- Select SSH-2 RSA or SSH-2 DSA from the parameters field.
  Note: The handshake with a DSA key is a bit faster.
- Click the generate button.
- Do not enter a passphrase. Server keys with passphrase are currently not supported.
- Export the key in the OpenSSH format.
- Transfer this key to the IPC@CHIP®.
- Specify the key filename in the **RSA_KEYFILE** or the **DSA_KEYFILE** entry, depending on your previous choice what key algorithm to use for server authentication.

Create a user, password:

- Specify a user name inside the **USERx** entry.
- Specify a password inside the **PASSWORDx** entry.

Create key for user public key authentication:

- Open 'puttygen.exe'.
- Select SSH-2 RSA or SSH-2 DSA from the parameters field.
- Click the generate button.
- Save the private key.
- Copy the text from the dialog 'Public key for pasting into OpenSSH authorized_keys file'.
- Past the text into a new text file.
- Get sure that there are no line breaks.
- If you like you can add other keys to this file. Each key must be in his own line.
- Transfer this file to the IPC@CHIP®.
- Specify the filename in the **USERx_KEYFILE** entry.
- Specify the private key filename inside your SSH client configuration.

Enable TCP Port Forwarding:

- By default the port forwarding option is disabled.
- You have to enable **LOCAL_FORWARD**, if you want to tunnel incomming TCP connections to a server located on the IPC@CHIP®.
- You have to enable **REMOTE_FORWARD**, if you want to tunnel outgoing TCP connections to a server outside the IPC@CHIP®.

---

**License Notes:**

The @CHIP-RTOS SSH Server implementation was ported from the **Dropbear** source code. Dropbear comes with the following license:


```
Copyright (c) 2002-2004 Matt Johnston
Portions copyright (c) 2004 Mihnea Stoenescu
All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the
Software without restriction, including without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to permit persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```


Key import code is modified from PuTTY's import.c, licensed as follows:


```
PuTTY is copyright 1997-2003 Simon Tatham.

Portions copyright Robert de Bath, Joris van Rantwijk, Delian Delchev, Andreas
Schultz, Jeroen Massar, Wez Furlong, Nicolas Barry, Justin Bradford, and CORE SDI
S.A.

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the
Software without restriction, including without limitation the rights to use, copy,
```

modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

---

End of document

---






# @CHIP-RTOS C Library - Helper Functions

---

## malloc

Reentrant malloc() for use with multitasking.

```
void * malloc (unsigned int size);
```

## Parameters

*size*

Number of bytes to allocate from heap.

## Return Value

Pointer to newly allocated memory on success
NULL pointer on allocation failure

## Comments

This function provides semaphore protection around the compiler's standard C-library malloc() function.

**This function is available only with the Paradigm Beck IPC Edition compiler.** The necessary hooks into the compiler's Run-Time Library (RTL) are not available with other compilers.

This function is not part of the Beck C-Library. You must link the provided HEAP.C source file directly into your project to use this reentrant heap manager. This source file can be found in the Paradigm Beck IPC Edition compiler's installation area in the Source/Beck directory.

The reentrant malloc(), **free()** and **realloc()** come together as one package inside the HEAP.C source file.

Refer to your compiler's standard library documentation for a description of this API's basic functionality.

**Caution:**

Here is a further argument for why Beck recommends large memory model for users who want their program to run right away.

For small or medium memory models, the task's stack must not only meet the DS == SS requirement, but also must have its stack located beyond the heap memory area in order to access the heap functions. Otherwise the `malloc()`call will fail due to a "heap level greater than SP" test that is performed down inside the RTL. (See `_RtlSbrk()`function in RTL's Memory/brk.cas source file for details.) So your only obvious option then would be to allocate task stack space from your DOS program's main stack for these memory models.

However, this will still leave you on shaky ground. The RTL's sense of "heap space exhausted" will not be reliable in these memory models, due to the mechanism cited above used for detecting

the upper bound of the heap area. Depending on which task is calling, the amount of available heap space will look different to the RTL. The 512 byte margin test used by the RTL between heap and SP stack pointer register makes sense for a single thread. But this test would would no longer be safe after you have sliced up the stack space into multiple stacks.

## See Also

**helper_alloc_rtos_mem** **free** **realloc**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Helper Functions

## free

Reentrant free() for use with multitasking.

```
void free (void * p);
```

## Parameters

*p*

Pointer to heap memory to be released.

## Return Value

-- none --

## Comments

This function provides semaphore protection around the compiler's standard C-library free() function.

**This function is available only with the Paradigm Beck IPC Edition compiler.** Refer to the reentrant **malloc()** description for more details on how to install this version of free() into your program.

Refer to your compiler's standard library documentation for a description of this API's basic functionality.

## See Also

**malloc** **realloc**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## realloc

Reentrant realloc() for use with multitasking.

```
void * realloc (void * p, unsigned int size);
```

## Parameters

*p*

Pointer to heap memory to be reallocated.

*size*

Number of bytes for new allocation.

## Return Value

Pointer to newly allocated memory on success
NULL pointer on allocation failure

## Comments

This function provides semaphore protection around the compiler's standard C-library realloc() function.

**This function is available only with the Paradigm Beck IPC Edition compiler.** Refer to the reentrant **malloc()** description for more details on how to install this version of realloc() into your program.

Refer to your compiler's standard library documentation for a description of this API's basic functionality.

## See Also

**malloc** **free**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

End of document




# @CHIP-RTOS C Library - Helper Functions

## helper_alloc_rtos_mem

Allocate memory from @CHIP-RTOS memory pool (Int21h function 48h)

```
unsigned int helper_alloc_rtos_mem(unsigned long size);
```

## Parameters

*size*

Requested memory size in bytes

## Return Value

If 0, memory allocation failed. Else segment of the allocated memory area

## Comments

The returned segment can be used to build a normalized pointer to the allocated memory area e.g.

```
seg=helper_alloc_rtos_mem(1024);
ptr = MK_FP(seg,0);
```

## See Also

helper_release_rtos_mem  helper_alloc_rtos_fast_mem  helper_assign_mem

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index




# @CHIP-RTOS C Library - Helper Functions

Header File: `#include <HELPER.H>`

## Char Character output Functions

| | |
|---|---|
| helper_strout | String output |
| helper_chrout | Char output |
| helper_dezout | Dezimal output |
| helper_hexdig | Hex digit output |
| helper_hexout | Hex value output |
| helper_printf | Printf output |
| helper_sprintf | String formatting |

## Mem Memory Allocation Functions

| | |
|---|---|
| helper_alloc_rtos_mem | Allocate RTOS memory |
| helper_release_rtos_mem | Release allocated RTOS memory |
| helper_alloc_rtos_fast_mem | Allocate RTOS memory from fast memory pool |
| helper_assign_mem | Assign RTOS memory ownership |
| --- Reentrant Heap --- | |
| malloc | Allocate program heap memory |
| free | Release program heap memory |
| realloc | Reallocate program heap memory |

## Other Other Functions

| | |
|---|---|
| helper_check_license | Check for installed license |
| helper_load_library | Load a Dynamic Link Library |
| helper_unload_library | Unload a Dynamic Link Library |

List of C Libraries
@CHIP-RTOS Main Index




# @CHIP-RTOS - BIOS Interface Documentation



## BIOS Interrupts

Here are the interface definition for the BIOS Interrupts

The system BIOS in a regular PC offers many services, only a subset of which is required in embedded systems. This subset is described here.

Unsupported BIOS functions are handled by a default handler which issues a message to the console.

Some additional functions not found in a normal PC BIOS are provided for your convenience.

For some useful comments see: **Programming notes**

BIOS API **Data Structures**

### BIOS Interrupt 0x10 Services

- **0x00:_Get char from standard input**
- **0x01:_Check if a character is available from std in**
- **0x08:_Read character at cursor**
- **0x09:_Write character at cursor**
- **0x0E:_Teletype output**
- **0x0F:_Get video state**
- **0x12:_Video subsystem configuration**

### BIOS Interrupt 0x11 Services

- **0xXX:_Get equipment list**

### BIOS Interrupt 0x16 Services

- **0x00:_Get char from standard input**
- **0x01:_Check if a character is available from std in**

### BIOS Interrupt 0x1A Services

- **0x00:_Get clock count since midnight**

## BIOS Interrupt 0xA0 Services


- 0x00: Get serial number
- 0x01: Get IP address of the Ethernet interface
- 0x02: Set IP address of the Ethernet interface
- 0x03: Get IP subnet mask of the Ethernet interface
- 0x04: Set IP subnet mask
- 0x05: Get IP gateway
- 0x06: Set IP default gateway
- 0x07: Execute a command shell command
- 0x08: Set timer 0x1C's interval
- 0x09: Set timer interrupt 0xAF's interval
- 0x0A: Get disk space info
- 0x11: Set STDIO focus
- 0x12: Get bootstrap version number
- 0x13: Get @CHIP-RTOS version number
- 0x14: Set batch file execution mode.
- 0x15: Continue batch file execution in BATCHMODE 1
- 0x16: Get information about the @CHIP-RTOS features
- 0x17: Get MAC address of the Ethernet interface
- 0x18: Power save
- 0x19: Change level for configuration server.
- 0x20: Install a user fatal error handler
- 0x21: Rebooting the IPC@CHIP®
- 0x22: Get version string
- 0x23: Insert an entry in chip.ini
- 0x24: Find an entry in chip.ini
- 0x25: Set the Stdio focus key
- 0x26: Get the IPC@CHIP® device name
- 0x27: Suspend/Resume System Servers
- 0x28: Fast Findfirst
- 0x29: Fast Findnext
- 0x2A: Long File Name (LFN) Findfirst
- 0x2B: Long file name (LFN) Findnext
- 0x2C: Long file name (LFN) FindDone
- 0x30: Fast Finddone
- 0x31: Detect Ethernet link state
- 0x32: Set a memory gap between the loaded DOS programs
- 0x33: Set stdin/stdout channel
- 0x34: Get stdin/stdout settings
- 0x35: Install user specific stdio handlers
- 0x36: Install a System Server Connection Handler function
- 0x37: Enable/Disable File sharing
- 0x38: Get file name by handle
- 0x3A: Get long file name by handle
- 0x40: Install a UDP Cfg Callback
- 0x45: Write persistent User Data
- 0x46: Read persistent User Data
- 0x48: Assign RTOS memory block to task
- 0x50: Get IP address of the PPP Server
- 0x55: Get IP address of the PPP Client
- 0x56: Get vsprintf address
- 0x57: Allocate Memory from fast memory pool

- 0x58: Get MAC address of the second Ethernet controller

---

## 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.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x11 service 0xXX: 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.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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, 8: User channel

### Comments

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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.)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x10 service 0x09: Write character at cursor

Write character to the standard output.

## Parameters

*AH*
Must be 0x09

*AL*
Character to write

## Return Value

Returns nothing.

## Comments

This function exists only for PC compatibility. At old DOS PCs this functions writes a character with specified attributes at the current cursor position. At the @CHIP-RTOS this function works in the same way than **service 0x0E**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.01 | V1.00 |

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.)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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, 3 in AL

mode currently display page ,0 in BH

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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, 8: User channel

## Comments

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Get the serial number of the IPC@CHIP® device

## Parameters

*AH*

Must be 0.

## Return Value

AX=low word, BX=high word of IPC@CHIP® serial number

CX=low word, DX=high word of BECK product serial number (if not available CX=DX=0)
SI contains BECK product hardware revision number HighByte and LowByte (if not available SI=0)
DI contains SCxxx hardware revision number HighByte and LowByte (if not available DI=0)

## Comments

The serial number is a 24 bit value.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Get the IPv4 address as a string.

## Parameters

*AH*
Must be 1.

*ES:DX*
Pointer to a 16 byte memory area where the IPv4 address is to be stored as a null terminated string.

## Related Topics

Ethernet **IPv4 address** initial value
Ethernet **IPv6 address** initial value
Convert ASCII IPv4 address to **binary**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

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

Set the Ethernet interface's IPv4 address based on the supplied string.

## Parameters

*AH*
Must be 2.

*ES:DX*
Pointer to a 16 byte memory area where the IPv4 address is stored as a null terminated string.

## Comments

This API is thread safe and reentrant.

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 IPv4 address **initial value**
Ethernet **IPv6 address** initial value
IPv4 address **IP** command line
Convert binary IPv4 address to **ASCII** dotted decimal

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Get the IPv4 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Set the IPv4 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

This API is thread safe and reentrant.

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Get the IPv4 gateway as a string.

## Parameters

*AH*
Must be 5.

*ES:DX*
Pointer to a 16 byte memory area where the IPv4 gateway is to be stored as a null terminated string.

## Related Topics

Ethernet default **gateway** initialization
**GATEWAY** command line
**ADD_DEFAULT_GATEWAY** API function
PPP server default **gateway** initialization

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

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

Set the IPv4 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

This API is thread safe and reentrant.

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`).

The TCP/IP 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

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.

## Return Value

AX==0 success
AX==-1 error (loading EXE file failed!)

## Comments

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Defines the interval in milliseconds for timer interrupt 0x1C.

## Parameters

*AH*
Must be 0x08.

*BX*
Interval in milliseconds

## Comments

Use `setvect(0x1c, my_function)` 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 C-library functions such as `printf`.

## Related Topics

chip.ini **TIMER 0x1C** configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Defines the interval in milliseconds for timer interrupt 0xAF.

### Parameters

*AH*
Must be 0x09.

*BX*
Interval in milliseconds

### Comments

Use `setvect(0xAF, my_function)` 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 C-library functions such as `printf`.

### Related Topics

chip.ini **TIMER 0xAF** configuration

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA0 service 0x0A: Get disk space info

Reports disk size and space.

### Parameters

*AH*
Must be 0x0A.

*AL*
Drive (0 current drive, 1=A, ...)

*ES:BX*

Pointer to **BiosDiskInfo** data structure containing disk drive statistics which will be filled in by this API call.

## Comments

This API is provided as an alternative to the DOS interrupt 0x21 service **0x36** "Get Disk Free Space" service. For large drives whose sector counts exceed 0xFFFF, this alternative API provided here should be used.

## Beck C-Library Usage

**BIOS_Disk_Info()** - Get disk information
**BIOS_Disk_Space()** - Get disk space

## Related Topics

DOS **service 0x36** Get Disk Free Space

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.16 | V1.00 |

**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 assure 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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.

## Related Topics

Get @CHIP-RTOS version number

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA0 service 0x13: Get @CHIP-RTOS version number

Get the version number of the @CHIP-RTOS.

## Parameters

*AH*

Must be 0x13

## Return Value

AX=version number, AH is major version, AL is minor version.
If DX is set it is a Beta version.

## Comments

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

## Related Topics

Get **bootstrap** version number
Get **version** string
Get **device** name

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

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

Sets the **batch file** execution mode of 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 in a batch file, 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 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 in a batch file waits forever for execution, if the predecessor program
does not finish or call **0xA0 Service 0x15**

## Related Topics

Initial **batch mode** configuration
**BATCHMODE** command

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x15: Continue 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x16: Get information about the @CHIP-RTOS features

Get information about running servers, interfaces and features of the @CHIP-RTOS

## 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 TCP/IP
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
Bit 8: IPv6 support
Bit 9: SSL Web server

BX:

Bit 0: SNMP MIB variables support (see **TCP/IP API service 0x71**)
Bit 1: UDP config server
Bit 2: Ping client (see **TCP/IP API service 0x75**)
Bit 3: TCP/IP device driver API(see **TCP/IP API service 0xA0 - 0xA7**)
Bit 4: Webfile upload (Webserver PUT method)

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 file transfer
Bit 5: External disk interface
Bit 6: Software SPI API
Bit 7: CAN-Bus API
Bit 8: USB API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## 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.

## Comments

At IPC@CHIP® targets without internal Ethernet (e.g. SC11), this number represents a virtual ID.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA0 service 0x18: Power save

**SC12:**

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.

**SC11/SC13/SC1x3/SC23:**

Slows down the processor clock and puts the CPU in a power save mode until the restore function is called or in halt mode until the next interrupt occurs. Please note that the internal time/date will be affected in some modes.

## Parameters

*AH*

Must be 0x18.

## SC11/SC13 Parameters

*AL*

Mode
If DX == 0xABCD and CX == 0x1234
  AL == 0x01: Restore full speed mode
  AL == 0x02: Set 1/2 speed mode
  AL == 0x04: Set 1/4 speed mode (serial baud rates > 57600 will no longer be accurate)
  AL == 0x80: Enter sleep mode (RTOS 1ms timer and time/date will be affected)
  AL == 0x81: Wakeup from sleep mode
  AL == 0x20: Power down Ethernet controller (no Ethernet communication is possible)
  AL == 0x21: Power up Ethernet controller
Else
  Put the CPU in halt mode until next interrupt occurs.
  Please note that the internal time/date will be affected.
  Call this function when your program is in idle state.
  Power savings are marginal.
Endif

*CX*

Set to 0x1234 if using AL as mode specifier

*DX*

Set to 0xABCD if using AL as mode specifier

## SC1x3/SC23 Parameters

*AL*

Mode
If DX == 0xABCD and CX == 0x1234
  AL == 0x01: Restore full speed mode
  AL == 0x02: Set 1/2 speed mode
  AL == 0x04: Set 1/4 speed mode
  AL == 0x20: Power down Ethernet controller (no Ethernet communication is possible)
  AL == 0x21: Power up Ethernet controller
  AL == 0x22: Disable the Ethernet Energy Detect Power-Down mode
  AL == 0x23: Enable the Ethernet Energy Detect Power-Down mode
Else
  Put the CPU in halt mode until next interrupt occurs.
  Please note that the internal time/date will be affected.
  Call this function when your program is in idle state.
  Power savings are marginal.
Endif

*CX*

Set to 0x1234 if using AL as mode specifier

*DX*

Set to 0xABCD if using AL as mode specifier

## SC11/SC13/SC1x3/SC23 Return Value

AX==0, if successful (only when DX==0xABCD and CX==0x1234)

## Comments

Please note that power consumption may differ slightly when the date code of the IPC@CHIP® is changed.

## SC12 Comments

Call this function when your program is in idle state.  Power savings are marginal since we use a DRAM.

## SC1x3/SC23 Comments

In mode 0x23 the Ethernet PHY is partially powered down.  The PHY powers-up when there is valid energy from the line: 100Base-TX, 10 Base-T or auto negotiation.
If the user calls this function with AL==4 (Set 1/4 speed mode), the internal ethernet controller doesn't work in the 100 MBit mode. Because of this behaviour, it is recommended to set the **ethernet mode** at chip.ini fix to 10MBit Halfduplex.

The modes 0x02 and 0x04 influence the USB API. In mode 0x02 the USB API cannot be used at all. The SC1X3 will crash if you do. In mode 0x04 the USB API can only be used in Device mode, because Host mode needs better performance.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## 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 **UDPCFG** documentation.

### 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.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Install a user fatal error callback function. This installed function will be called by the system on execution of fatal errors.

### Parameters

*AH*
Must be 0x20.

*ES:DI*
Address of the user error handler function (or zero to remove a previously installed handler).

### Return Value

ES:DI holds old handler (since SC1x RTOS V1.25 / SC1x3/SC23 RTOS V1.20)

## Comments

The user is permitted to execute an error handler function if a fatal error occurs in either an application program or the @CHIP-RTOS.

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

Only one error handler callback per system is supported. An application can remove its handler by calling this install function with a zero (NULL) value in ES:DI, which is an advisable clean-up procedure at program exit.

The callback function must be of type `huge _pascal` with an `errorcode` input parameter (see the example function below). The error codes conveyed by this input parameter are as follows:

01: Invalid processor opcode (usually caused by corrupted memory). The current task will be suspended.
02: Fatal kernel error (usually caused by corrupted memory or a task stack overflow)
03: Fatal internal TCP/IP error, current task will be suspended
04: TCP/IP stack reaches memory limit
05: TCP/IP memory allocation error
07: Ethernet link error detected (SC13 only: cable not connected?)
08: Flash write error -> Flash defect ( **Note:** IPC@CHIP® is no longer usable)
09: Low Memory error -> called if a memory allocation (system or user) failed.
10: RTOS memory list corrupted

In all cases (except errorcode 7 and 9) we recommend a reboot with BIOS interrupt 0xA0 **0x21**.

**Important:** Do not use any message printing inside your error handler if error code is 3 or 4, because your exit handler will hang inside the print call when Telnet is part of your stdio.

```
// Example using Beck C-Library functions:
//   This error handler reboots the system
void huge _pascal user_error_handler(int errorcode)
{
   BIOS_Reboot () ;
}

// Installing the error handler function
//  from program's main function
BIOS_Install_Error_Handler (user_error_handler) ;
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.20 | V1.20 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x21: Rebooting the IPC@CHIP®

This function works in the same way as the **reboot** shell command

## Parameters

*AH*
0x21

## Return Value

No return from this function occurs due to system reboot.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

Copies the @CHIP-RTOS 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

Get version **number**
Get **device** name
**VER** shell command

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

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

The functions 0x23 and 0x24 allows the user to modify/place and find/read your 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. 128 chars)
Pass a NULL pointer here to erase the complete item line

## Return Value

AX=0 success
AX=-1 Invalid string length, low heap memory or file error.

## Comments

This API is thread safe and reentrant. This function will fail (no operation) if insufficient system heap memory is available to momentarily hold the contents of the CHIP.INI file.

This API can be accessed through the Beck C-Library function **BIOS_Set_Ini_String**. A call to this function with parameters:

```
BIOS_Set_Ini_String("MY_SECTION", "MY_ITEM", "VALUE_TEXT");
```

produces the following chip.ini entry:

```
[MY_SECTION]
MY_ITEM=VALUE_TEXT
```

Keep in mind that this function writes to the chip.ini file. This generates flash write cycles and these cycles are limited.

## Related Topics

**Get** CHIP.INI entry

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

Finds an entry in chip.ini configuration file.

## Parameters

*AH*
Must be 0x24.

*CX*
Maximum length of target string (without '\0').

*BX:SI*
Pointer to section string

*ES:DI*
Pointer to item name

*DS:DX*
Pointer to target

## Return Value

AX= 0 : Entry not found
AX= -1: Could not open chip.ini or low memory
else Success: pointer at DS:DX contains the found string AX contains length of the found string

## Comments

Leading and trailing white space characters will be cut. The function stops reading the entry when it finds an invalid character (<0x20).
This API is thread safe and reentrant. This function will fail if insufficient system heap memory is available to momentarily hold the contents of the CHIP.INI file.

The maximum length value in CX specifies the number of characters which could be copied into the target string. You have to allocate one more character for the null termination.

The Beck C-Library function **BIOS_Get_Ini_String** can be used to access this API. Example usage would be as follows:

```
// Declare a target buffer to be filled by BIOS_Get_Ini_String().
unsigned char target[101];

BIOS_Get_Ini_String("MY_SECTION", "MY_ITEM", target, 100);
// Now target contains the chip.ini text for this item
```

## Related Topics

**Set** CHIP.INI entry

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xA0 service 0x25: Set the Stdio focus key

Set the Stdio focus key

### Parameters

*AH*
Must be 0x25.

*AL*
Focus key 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xA0 service 0x26: Get the IPC@CHIP® device name

Get the IPC@CHIP® device name

### Parameters

*AH*
Must be 0x26.

### Return Value

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

DX:CX contains pointer to the fixed BECK product name stored in the IPC@CHIP® flash

## Comments

All returned strings are terminated by 0. These strings should be treated as read only.

## Related Topics

**Device name** definition
Get version **number**
Get version **string**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**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

## SC1x Parameters

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

## SC1x3/SC23 Parameters

*BX*
0: FTP Server, 1: Telnet Server, 2: Web Server, 3: SSL Web Server, 4: SSH Server

## Return Value

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

## Comments

If FTP, Telnet or WEB was disabled at startup with a CHIP.INI entry, you can enable it using this call.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA0 service 0x28: Fast Findfirst

Provide a an access to the file system directories which locks the directory against write activity during the series of search calls.

## Parameters

*AH*
Must be 0x28.

*CX*
File attribute (provisional, not used)

*BX:SI*
Null terminated file specification

*ES:DI*
Pointer to a **filefind** structure which will be initialized inside this API.

## Return Value

AX= 1 DX=0: Success
AX= 0 DX=0: No file found

## Comments

The three filefind functions (0x28, 0x29, 0x30) provide a slightly faster Findfirst/next access than the DOS compatible functions at INT **21h**. More importantly, these functions provide a multi-tasking version of the findfirst/findnext functions which protects against changes being made to the file directory within the midst of the search sequence. The directory being searched will remain locked against write activity until the search is completed (Findfirst/Findnext return with AX equal zero) or until the **Finddone** API terminates the search.

This function starts an iteration of files found at the path specified in [BX:SI]. This path may contain the * and ? wild card characters. If this function returns with AX set to 1, it has successfully found a matching file at the specified path. The **Findnext** function can then be used to retrieve further matching file names.

This function is reentrant. However, eventually the system will run out of file system resources (e.g. directory nodes) if too many sessions are simultaneously active.

**CAUTION:**

The **Fast Finddone** function must be called if the Findfirst/Findnext sequence is stopped prior to these iteration functions returning with AX equal zero ("File not found"). This is important so that the directory can be unlocked and file system resources released!

## SC1x3/SC23 Comments

The alternative directory scan functions for long file names (LFN) starting with **LFN Findfirst** can be used to read out the long file names.

## Related Topics

**Fast Findnext**
**Fast Finddone** must be called at end of the search
**LFN Findfirst**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

**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** structure initialized by the **Fast Findfirst** function (0x28).

## Return Value

AL= 1 DX=0: Success
AX= 0 DX=0: no file found

## Comments

This function is reentrant.

See **Fast Findfirst function (0x28)** for further description.

**CAUTION:**

The **Fast Finddone** function must be called if the Findfirst/Findnext sequence is stopped prior to these iteration functions returning with AX equal zero ("File not found"). This is important so that the directory can be unlocked and file system resources released!

## Related Topics

**Fast Findfirst** must be called to start the search
**Fast Finddone** called at end of the search

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x2A: Long File Name (LFN) Findfirst

Begin iteration of files listed in specified directory.

## Parameters

*AH*
Must be 0x2A.

*AL*
non-zero to leave the directory node locked against write activity

*BX:SI*
Null terminated file path

*ES:DI*
Pointer to a **LFN_filefind** structure which will be initialized inside this API.

## Return Value

CX= 1: Success
CX= 0: No file found
Carry flag zero (in either case)

## Comments

The set of long file name file find functions (0x2A, **0x2B** and **0x2C**) provide access to the long file names as well as the short [8.3] DOS alias names. These functions operate roughly similar to the DOS compatible functions at INT **21h** when AL parameter is zero. No directory node lock is performed.

When AL parameter is non-zero, then these functions behave analogous to the so called Fast findfirst BIOS service **0x28** but providing the long file names. In this case these functions lock the file node (write protect) of the directory being scanned and therefore are protected against directory changes being made during the iteration sequence. This lock remains in affect until either a "No file found" indication is returned from either this function or Findnext (0x2B), or until the user calls the LFN FindDone API, **0x2C**, to terminate the search.

This function starts an iteration of files found at the path specified in [BX:SI]. This path may contain the * and ? wild card characters. If this function returns with AX set to 1, it has successfully found a matching file at the specified path. The **LFN Findnext** function can then be used to retrieve further matching file names.

This function is reentrant. However, eventually the system will run out of file system resources (e.g. directory nodes) if too many sessions are simultaneously active. This resource exhaustion with excessive recursion is more likely when the AL non-zero node locking is used.

Note: The alternative **Fast Findfirst** / **Fast Findnext** functions which retrieve only the short alias DOS file names are no faster than this series of long name findfirst/findnext functions. (Those functions are named "fast" only for historical reasons.)

**CAUTION:**

System heap memory, and file system resources remain allocated until the user calls LFN FindDone (**0x2C**) or until either of the LFN FindFirst/FindNext functions return with CX equal zero. Therefore it is important that the user call the LFN FindDone API if the search is terminated before the FindFirst/FindNext functions reach the "No file found" condition. Calling the LFN FindDone function for "good measure" after the FindFirst/FindNext functions have returned the zero in CX does no harm (has no affect).

## Related Topics

**Long File Name Findnext**
**Long File Name Finddone**
Short name **"Fast" Findfirst**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x2B: Long file name (LFN) Findnext

Continues a file search which was started by LFN Findfirst (0x2A)

## Parameters

*AH*
Must be 0x2B.

*ES:DI*
Pointer to **LFN_filefind** structure initialized by **LFN Findfirst** call.

## Return Value

CX= 1: Success
CX= 0: no file found

## Comments

This function is reentrant.

See **LFN Findfirst** function (0x2A) for further description.

**CAUTION:**

System heap memory, and file system resources remain allocated until the user calls LFN FindDone (**0x2C**) or until either of the LFN FindFirst/FindNext functions return with CX equal zero. Therefore it is important that the user call the LFN FindDone API if the search is terminated before the FindFirst/FindNext functions reach the "No file found" condition.

## Related Topics

**LFN Findfirst** must be called to start the search
**LFN Finddone** must be called at end of the search

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x2C: Long file name (LFN) FindDone

Terminates a search which was started by LFN Findfirst (0x2A), releasing resources.

## Parameters

*AH*
Must be 0x2C.

*ES:DI*
Pointer to **LFN_filefind** structure initialized by **LFN Findfirst** call.

## Return Value

Carry flag zero

## Comments

This function releases the system resources required during a directory scan operation.

This function is reentrant.

See **LFN Findfirst** function (0x2A) for further description.

**CAUTION:**

System heap memory, and file system resources remain allocated until either the user calls LFN FindDone or either the LFN **FindFirst**/**FindNext** functions return with CX equal zero. Therefore it is important that the user call this LFN FindDone API if the search is terminated before the FindFirst/FindNext functions reach the "No file found" condition.

## Related Topics

**LFN Findfirst** must be called to start the search

LFN Findnext continues search

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA0 service 0x30: Fast Finddone

Closes a find access started with Fast Findfirst, thereby releasing the lock on the directory that was being searched.

## Parameters

*AH*

Must be 0x30.

*ES:DI*

Pointer to filefind structure initialized by the Fast Findfirst function (0x28).

## Return Value

AX=0 DX=0: Success

## Comments

This function releases the file system resources used during a directory scan operation.

This function is reentrant.

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

## Related Topics

Fast Findfirst
Fast Findnext

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

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?)
DX!=0: Initialization or reset procedure of Ethernet device failed

## SC12 Return Value

CX=0

## SC13/SC1x3 Return Value

CX holds the current Phy status.

Bit 14 1 = Link Not detected
Bit 07 1 = 100Base-TX mode, 0=10Base-T mode
Bit 06 1 = Device in Full Duplex mode

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

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

Sets a memory gap between loaded DOS programs as a memory reserve.

## 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 found in BX

## Comments

Some programs compiled with Borland C 5.02, Paradigm Beck Edition (other compilers??) try to increase their program memory block at runtime. This can occur, for example, when opening a file with the Borland or Paradigm C-library function `fopen`, where some additional memory is required. The C-library `fopen` function calls int 21h **0x4A**, which is not directly visible to the application programmer. This memory resize call fails if another program is loaded after the previous one, because now there is no memory space left for increasing the memory size of the previously executed program. The program then returns from `fopen` with an error. In this case, the global program variable errno is set to value 8 (not enough memory).

To prevent this error, the @CHIP-RTOS allows a memory gap of a defined size between loaded programs. This memory gap size is specified as a number of paragraphs (where 1 paragraph equals 16 bytes for SC1x, and 256 bytes for SC1x3 systems).

This value can also be defined in **chip.ini**.

Note: This strategy can fail when programs are terminated and restarted again.
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, should this value be defined. We recommend in that case a value of 128 paragraphs for SC1x or 8 paragraphs for SC1x3 (2048 bytes in either case). The described problem was noticed when the Borland or Paradigm C-library function `fopen` were used. The same can happen with usage of C-library function `malloc` using memory model Large. The `malloc` returns a NULL pointer in this case.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

**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
Bit 4: SSH channel (SC1x3 only)

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V1.07 | V1.00 |

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
Bit 4 == 1: SSH channel (SC1x3 only)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V1.07 | V1.00 |

Top of list
Index page

---

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

Installs user specific stdio channel handlers.

This API call allows the user to install their own stdio handler functions, e.g. for a user developed input/output device connected to the IPC@CHIP® (e.g. a display and/or keyboard device) or a user TCP application similar to Telnet.

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

After installing these functions with this API call and setting stdin and/or stdout to include the user channel with **Set stdio channels**, the @CHIP-RTOS will call these user functions at each stdin and stdout operation. For further details on programming and installing these handler functions, see the comments below.

## Parameters

*AH*

Must be 0x35.

*ES:DI*

Pointer to User_Stdio_Funcs structure variable, see comments

## 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 be implemented 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 four callback function vectors in their User_Stdio_Funcs type variable and then call this API function with the address of the User_Stdio_Funcs structure in ES:DI.

With ES = DI = 0, the user can later uninstall their stdio handlers.

**Important:**

1. If your applications exits, don't forget to uninstall your stdio handlers, by calling this API call with ES=DI=0.

2. Do not call your implemented stdio functions from inside your application. These installed functions must be called only from inside the @CHIP-RTOS.

This API can be reached using the `BIOS_Install_User_Stdio` Beck C-Library function. Example usage of this function is shown below.

```
User_Stdio_Funcs user_stdio_funcs;    // global variable

// Implementation of users stdio functions
int  huge my_kbhit(void)
{
  //........
}

int huge my_getch(void)
{
  //........
}

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

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

void install_mystdio_channel(void)
{
  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;

  BIOS_Install_User_Stdio ( &user_stdio ) ;
  BIOS_Select_Stdio ( STDIO_COM | STDIO_TELNET | STDIO_USER,
           SET_STDOUT | SET_STDIN) ;
}

void remove_mystdio_channel(void)
{
  BIOS_Select_Stdio ( STDIO_COM | STDIO_TELNET,
           SET_STDOUT | SET_STDIN) ;

  BIOS_Install_User_Stdio ( 0 ) ;
}

int main(void)
{
  //......
  install_mystdio_channel();
  //....

  // at the end of the program
  remove_my_stdio_channel():
}
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

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

Installs 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 their own callback functions for controlling access to the default @CHIP-RTOS servers.

## Parameters

*AH*
Must be 0x36.

*ES:DI*
Pointer to handler function

## SC1x Parameters

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

## SC1x3/SC23 Parameters

*BX*
0: FTP Server, 1: Telnet Server, 2: Web Server, 3: SSL Web Server, 4: SSH Server

## Return Value

AX=0: Success
AX=-1: Invalid Parameter

## Comments

The connection handler will be called when a client establishes a connection to the server (FTP, WEB, Telnet).
Exemplary usage:
The implemented handler function can check the source IP address (Clients IP), compare this IP with an application internal list of allowed IP addresses. The connection can then be rejected by returning a non-zero value if the source IP is not a member of the list.

A connection handler function must be declared in the following manner:

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

If the handler returns 0 the connection will be established. If it returns a nonzero value, the connection will be aborted.
To uninstall a connection handler call this function with a null pointer (ES=DI=0).

## SC1x Comments

Example handler:
The handler can read the IP Address and the Port in the **sockaddr_in struct** . (If desired, the IP address in the `sin_addr` structure member can be converted to ASCII using the TCP/IP **API_INETTOASCII** function.)

```
int huge UserConnectionHandler( struct sockaddr * SockAddr )
```

```
{
    char ip_str[INET_ADDRSTRLEN];
    InetToAscii(&((struct sockaddr_in *)SockAddr)->sin_addr.s_addr
                ,ip_str);
    helper_printf("\r\nHost %s, Port %u",ip_str,
                ((struct sockaddr_in *)SockAddr)->sin_port);
    return 0;
}
```

## SC1x3/SC23 Comments

Example handler:
The handler can read the IP Address and the Port in the **sockaddr_storage struct** .

```
int huge UserConnectionHandler( struct sockaddr * SockAddr )
{
  char ip_str[INET6_ADDRSTRLEN];
  struct sockaddr_in ipv4SockAddr;

  if(IN6_IS_ADDR_V4MAPPED(
       &((struct sockaddr_storage *)SockAddr)->addr.ipv6.sin6_addr))
  {
   //IPv4, copy data into  sockaddr_in variable
   ipv4SockAddr.sin_addr.s_addr=((struct sockaddr_storage *)SockAddr)
                               ->addr.ipv6.sin6_addr.ip6Addr.ip6U32[3];
   ipv4SockAddr.sin_port=
         ((struct sockaddr_storage *)SockAddr)->addr.ipv6.sin6_port;
   //Convert to string
   inet_ntop( AF_INET,&ipv4SockAddr.sin_addr,ip_str,INET_ADDRSTRLEN);

   //helper_printf(.........);
  }
  else
  {
    //IPv6: //Convert to string
    inet_ntop( AF_INET6,
              &((struct sockaddr_storage *)SockAddr)
              ->addr.ipv6.sin6_addr,ip_str,INET6_ADDRSTRLEN);
    //helper_printf(...........);
  }
  return 0;
}
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x37: Enable/Disable File sharing

Enable disable File sharing 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

By default file sharing 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 only be opened for read access a further time.

To avoid this security feature you can enable file sharing. This can also be done with the CHIP.INI entry **FILESHARING**.

NOTE: Be careful when opening a file multiple times with one or more of the openings done with write access!

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x38: Get file name by handle

Returns the file name string corresponding to a specified file handle.

## Parameters

*AH*
Must be 0x38.

*CX*
File handle

*ES:BX*
Output Parameter: Pointer to string, must be 13 chars long (12 + Null termination)

## Return Value

AX=0: Success
AX=-1: Invalid file handle

## Comments

This function is reentrant.

## SC1x3/SC23 Comments

The alternative **Get LFN** function can be used to retrieve the long file names.

## Related Topics

**Get LFN**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x3A: Get long file name by handle

Returns the long file name string corresponding to a specified file handle.

## Parameters

*AH*
Must be 0x39.

*CX*
File handle

*ES:BX*
Output parameter: Pointer to buffer in user memory where up to 259 ASCII characters plus a null terminator will be written.

## Return Value

AX=0: Success
AX=-1: Invalid file handle

## Comments

On success, the long file name is written to the caller's buffer at [ES:BX].

This function is reentrant.

## Related Topics

**Get DOS** file name

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA0 service 0x40: Install a UDP Cfg Callback

Install a UDP Config Server User Callback function.

## Parameters

*AH*
Must be 0x40.

*ES:DI*
Pointer to the User Callback function

## Comments

This API Function installs a User Callback function for the UDP Config Server. If a UDP Cfg Request with the command number 06 arrives, this User Callback function will be called. This allows you to realize your own UDP Config sub protocol and commands. The UDP Cfg Callback function receives an argument with information about the UDP Cfg Request and its requester. This information structure and the function must be declared as specified in the description below:

```
typedef struct UdpCfgSrv_UserCBInfo
{
    int length;                     // Length of this struct
    struct sockaddr_in *fromAddrPtr;    // Sender address pointer
    int udpCfgSD;                // UDP Config Server's Socket Descriptor
    char *dataPtr;               // Data of Request package
    unsigned dataLength;         // Length of request package
};

void huge MyUdpCfgSrvCB( struct UdpCfgSrv_UserCBInfo *infoPtr );
```

If the callback function returns with the `dataPtr` and `dataLength` fields in the `UdpCfgSrv_UserCBInfo` structure both non-zero, the UDP Config Server will send the `dataLength` bytes from location referenced by `dataPtr` back to the requester. If the pointer is set to null or the `dataLength` field is set to 0, no data will be sent back to the requester.

To remove an installed callback function, call this function with a null pointer.

For more Information on the UDP Config Server and its protocol, refer to the UDP Config Server description available on our website.

Note: The data sent and received is limited to 300 bytes maximum.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list

Index page

## Interrupt 0xA0 service 0x45: Write persistent User Data

Writes persistent data into the Flash memory of the @CHIP

### Parameters

*AH*
Must be 0x45.

*ES:SI*
Pointer to the User data

*CL*
number of bytes to write (max 192)

### Comments

The user can use this function to save product specific persistent data (e.g. own serial number of the product).
The function requires a pointer in [es:si] to memory block, which should be written into the flash memory.
The 192 bytes array in the flash memory will be untouched on a format of the file system and also on an @CHIP-RTOS BIOS Update.

### Related Topics

Read persistent User Data

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xA0 service 0x46: Read persistent User Data

Reads persistent data from the Flash memory of the @CHIP

### Parameters

*AH*
Must be 0x46.

*ES:DI*
Pointer to the User buffer

*CL*
number of bytes to read (max 192)

### Comments

The user can use this function to read product specific persistent data written with function **0x45** (Write Persistent User Data).
The function requires a pointer [es:di] to the memory block into which the read data will be stored.

## Related Topics

**Write** persistent User Data

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x48: Assign RTOS memory block to task

The indicated memory block is re-assigned to the specified RTOS task.

## Parameters

*AH*
Must be 0x48.

*ES:BX*
Pointer to the memory block allocated by the RTOS.

*CX*
RTOS **task ID** of task to assign this memory block. You may set this to zero to assign the memory block to the current task that is making this call.

## Return Value

Success: AX = 0
Invalid task ID: AX = -1
Invalid RTOS memory pointer: AX = -38

## Comments

This function can be useful to control the lifespan of a given block of memory. After the specified task has been assigned ownership of this memory block, this memory block will be automatically released when that task is **deleted** from the system if the block had not already been released.

A task's stack space allocated from system memory can safely be released in this manner.

## Related Topics

Allocate **normal** RTOS memory
Allocate **fast** RTOS memory
**Delete** task

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.12 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA0 service 0x50: Get IP address of the PPP Server

Get the IP address as a string.

## Parameters

*AH*

Must be 0x50.

*ES:DX*

Pointer to a 16 byte memory area where the IP address is to be stored as a null terminated string. If no PPP IP address is set (e.g. no link is established) this function returns an empty string.

## SC1x3/SC23 Comments

This function is deprecated, because it doesn't work, if the PPP server uses the IPv6 protocol. The IP configuration of all TCPIP/IP devices can be read by using **Get_Iface_Entries** or **Get_IPv6_Iface_Entries**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA0 service 0x55: Get IP address of the PPP Client

Get the IP address as a string.

## Parameters

*AH*

Must be 0x55.

*ES:DX*

Pointer to a 16 byte memory area where the IP address is to be stored as a null terminated string. If no PPP IP address is set (e.g. no link is established) this function returns an empty string.

## SC1x3/SC23 Comments

This function is deprecated, because it doesn't work, if the PPP server uses the IPv6 protocol. The IP configuration of all TCPIP/IP devices can be read by using **Get_Iface_Entries** or **Get_IPv6_Iface_Entries**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x56: Get vsprintf address

Get the address of the internal RTOS `vsprintf` function.

## Parameters

*AH*

Must be 0x56.

## Return Value

AX=0
ES:DI points to the internal `vsprintf` function of the @CHIP-RTOS BX = 0x1 to signal caller that this @Chip-RTOS version supports the direct console output when first argument to the `vsprintf` function is MK_FP(0, 0x22). (If BX remains unchanged, then @Chip-RTOS version is older and does not support this feature.)

## Comments

This function is used by our C-library to provide a `printf` function for user applications. Floating point types are not supported by this `vsprintf` implementation.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x57: Allocate Memory from fast memory pool

Allocates memory for the calling process from the fast memory pool.

## Parameters

*AH*

0x57

*BX*
Size counted in paragraphs

## Return Value

Success:
AX holds the segment of the memory area
Failure:
AX = 0, due to not enough memory available.

## Comments

A paragraph is 256 bytes in length.

**Note:**

The software Cryptographic Engine used by **SSH**, **SSL** or **IPsec** will operate noticeably quicker during key negotiation (e.g. Diffie-Hellmann, RSA, DSS) if a block of memory in the fast memory pool is available. A memory block of roughly 42 Kbyte size is required by the Cryptographic Engine. If the block is not available in this fast RAM area, then an attempt is made to allocate it from the normal system memory as a last resort.

## Beck C-Library Usage

**helper_alloc_rtos_fast_mem()**

## Related Topics

Allocate **normal** RTOS memory
**Free** allocated memory
**Reassign** memory ownership

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA0 service 0x58: Get MAC address of the second Ethernet controller

Get the MAC address as a 6 Byte array.

## Parameters

*AH*
Must be 0x58.

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

## Comments

The second Ethernet controller has no PHY controller. Only an MII interface is provided. It is only accessible with an external connected PHY and an external driver program, which must base on the **TCPIP device driver API**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | n/a |

**Top of list**
**Index page**

End of document




# @CHIP-RTOS - TCP/IP Application Programmer's Interface



## TCP/IP API

TCP/IP API

This interface provides access to the IPC@CHIP® TCP/IP stack's socket interface for programming TCP/IP applications. The TCP/IP API are all reached though software interrupt 0xAC. The desired service is selected with the high order byte of the AX register (AH).

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

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

For some useful comments see **Programming notes**.

All needed constants (e.g. API_OPENSOCKET) and data structures are defined in the header file tcpipapi.h.

TCP/IP API **Error Codes** Listing
TCP/IP API **Developer** Notes
TCP/IP API **Data Structures**
TCP/IP API **TCP/IP IPv4/v6 Dual layer stack**
TCP/IP API **Client/Server applications**
TCP/IP API **IP Security Data Structures**

**Notes** :

- "Network byte order" is big endian (like Motorola machines, unlike Intel), where most significant byte appears at the lowest address byte of a multibyte integer.
- 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

## TCP/IP Interrupt 0xAC Services

Service is selected by index in AH register.

## Common TCP/IP Socket functions

- 0x00: _Install dynamic link, then call function
- 0x01: _API_OPENSOCKET, open a socket
- 0x02: _API_CLOSESOCKET, close a socket
- 0x03: _API_BIND, bind TCP or UDP server socket
- 0x04: _API_CONNECT, Connect to another socket
- 0x05: _API_RECVFROM, Receive UDP message
- 0x06: _API_SENDTO, Transmit a UDP datagram
- 0x07: _API_HTONS, Convert byte order
- 0x08: _API_INETADDR, Convert an IP-String to binary
- 0x0C: _API_GETRCV_BYTES, Get waiting bytes count
- 0x0D: _API_ACCEPT, Accept the next incoming connection
- 0x0E: _API_LISTEN, Listen for incoming connections
- 0x0F: _API_SEND, Transmit TCP message
- 0x10: _API_RECV, Receive TCP message
- 0x11: _API_INETTOASCII, Convert IP address to IP string
- 0x12: _API_RESETCONNECTION, Abort a connection on a socket
- 0x13: _API_SETLINGER, Set linger time on close
- 0x14: _API_SETREUSE, Set reuse option on a listening socket
- 0x15: _API_SETIPTOS, Set IP Type-Of-Service
- 0x16: _API_SETSOCKOPT, Set options on socket
- 0x17: _API_GETSOCKOPT, Get options on socket
- 0x18: _API_SETBLOCKINGMODE, Set socket mode
- 0x19: _API_REGISTERCALLBACK, Register a user callback function
- 0x1A: _API_SELECT, Wait on socket events
- 0x20: _API_REGISTERCALLBACK_PASCAL, Pascal user callback
- 0x21: _API_GET_SOCKET_ERROR, Get last socket error.
- 0x22: _API_GET_TCP_STATE, Find TCP socket and return state.
- 0x23: _API_FINDALL_SOCKETS, Get information about all open sockets.
- 0x24: _API_GET_TCP_STATE_IP_ALL, Find TCP sockets and return state
- 0x25: _API_FINDALL_SOCKETS_IP_ALL, Get information about all open sockets.
- 0x26: _API_SHUTDOWN, Shutdown socket.
- 0x27: _API_RECVFROM_TO, Receive UDP message
- 0x28: _API_SENDTO_IFACE, Transmit a UDP datagram via specified interface
- 0x7A: _IP_USER_CB, Install IP callback function
- 0x7B: _ARP_USER_CB, Install ARP callback function
- 0x7C: _IP_USER_CB_SND, Install IP callback function on outgoing packets
- 0xC1: _INET_PTON, Convert an IP address text to binary
- 0xC2: _INET_NTOP, Convert an IP binary address to text
- 0xC3: _API_OPENSOCKET_IPV6, open an IPv6 socket

## DNS services

- 0x30: _DNS_GET_HOST_BY_NAME, Get a host's address(es) by domain name (IPv4)
- 0x31: _DNS_GET_HOST_BY_NAME2, Get a host's address(es) by domain name (IPv6)
- 0x32: _DNS_GET_HOST_BY_ADDR, Get a host's domain name by address (IPv4)
- 0x33: _DNS_GET_HOST_BY_ADDR2, Get a host's domain name by address (IPv6)
- 0x34: _DNS_GET_MAIL_HOSTS, Get mail hosts
- 0x35: _DNS_SET_NAME_SERVER, Add prioritised name server

## PPP client functions

- 0x40: _PPPCLIENT_INSTALLED, Check if PPP client installed.
- 0x41: _PPPCLIENT_OPEN, Open a PPP connection

- 0x46: _PPPCLIENT_OPEN_IPV6, Open a PPP connection
- 0x42: _PPPCLIENT_CLOSE, Close a PPP client connection
- 0x43: _PPPCLIENT_GET_STATUS, Get PPP client status
- 0x44: _PPPCLIENT_GET_DNSIP, Get DNS IP address
- 0x45: _PPPCLIENT_SET_OPTIONS, Set options for the PPP client

### PPP server functions

- 0x50: _PPPSERVER_INSTALLED, Check if PPP server installed
- 0x51: _PPPSERVER_SUSPEND, Suspend PPP server task
- 0x52: _PPPSERVER_ACTIVATE, Activate PPP server
- 0x53: _PPPSERVER_GET_STATUS, Get server state
- 0x54: _PPPSERVER_GET_CFG, Get PPP server configuration
- 0x55: _PPPSERVER_SET_OPTIONS, Set options for PPP server
- 0x56: _PPPSERVER_GET_CFG_IPv6, Get PPP server configuration

### DHCP

- 0x72: _DHCP_USE, Enable/Disable DHCP usage
- 0x73: _DHCP_STAT, Get DHCP status of the Ethernet interface

### Ping services

- 0x75: _PING_OPEN, Open and start ICMP echo requests
- 0x76: _PING_CLOSE, Finish ICMP echo requests
- 0x77: _PING_STATISTICS, Retrieve ping information

### ARP/Routing

- 0x80: _ADD_DEFAULT_GATEWAY, Add the default gateway
- 0x81: _DEL_DEFAULT_GATEWAY, Delete the default gateway
- 0x82: _GET_DEFAULT_GATEWAY, Get the current default gateway
- 0x83: _ADD_STATIC_ROUTE, Add a route for an interface
- 0x84: _DEL_STATIC_ROUTE, Delete a route for an interface
- 0x88: _DEL_ARP_ENTRY_BY_PHYS, Delete by physical address
- 0x89: _ADD_ARP_ENTRY, Add an entry to the ARP table
- 0x8A: _GET_ARPROUTE_CACHE, Read ARP/Route cache table

### IP Multicast

- 0x90: _ADD_IGMP_MEMBERSHIP, Install an IP multicast address entry
- 0x91: _DROP_IGMP_MEMBERSHIP, Delete an IP multicast address entry
- 0x92: _MCASTIP_TO_MACADDR, Map IP multicast address to Ethernet

### Device driver API

- 0x9F: _DEV_GET_IP, Get IP address and net mask of user device
- 0xA0: _DEV_OPEN_IFACE, Install user device driver

- 0xA1: _DEV_CLOSE_IFACE, Close TCP/IP device driver/interface
- 0xA2: _DEV_RECV_IFACE, Move received data
- 0xA3: _DEV_RECV_WAIT, Wait for received data
- 0xA4: _DEV_NOTIFY_ISR, Signal from ISR
- 0xA5: _DEV_GET_BUF, Get a buffer from TCP/IP stack
- 0xA6: _DEV_SND_COMPLETE, Signal message send complete
- 0xA7: _DEV_WAIT_DHCP_COMPLETE, Wait for DHCP
- 0xA8: _DEV_PPP_REGISTER_CHAP, Install a CHAP authentication callback function.
- 0xA9: _DEV_PPP_REGISTER_PAP, Install a PAP authentication callback function
- 0xAA: _DEV_PPP_SET_OPTION, Set PPP options
- 0xAB: _DEV_PPP_GET_PEER_IP, Get PPP peer IPv4 address
- 0xAC: _DEV_PPP_GET_PEER_DNS, Get PPP peer address
- 0xAD: _DEV_GET_HANDLE_BYNAME, Get driver handle pointer
- 0xAE: _DEV_CONFIG_IFACE, Configure an interface
- 0xAF: _DEV_CONFIG_IFACE_IPV6, IPv6 configuration of an interface
- 0xB0: _DEV_UNCONFIG_IFACE, Unconfigure an interface
- 0xB1: _DEV_PPP_GET_PEER_IPV6, Get PPP peer IPv6 address

## IPsec/PKI/SSL functions

- 0xCF: _IPSEC_VECTORS, IP Security API
- 0xD0: _PKI_CERT_ADD, Add a certificate
- 0xD1: _PKI_CERT_DEL, Delete a certificate
- 0xD2: _PKI_OWNKEYPAIR_ADD, Add a public/private key pair
- 0xD5: _SSL_GET_CONNSTATE, Return internal SSL status information
- 0xD6: _SSL_HANDSHAKE_COMPLETE, Test SSL handshake status
- 0xD7: _SSL_SESSION_CLOSE, Close a SSL session
- 0xD8: _SSL_SESSION_NEW, Open a new SSL session
- 0xD9: _SSL_SESSION_PROP, Set proposals for a SSL session
- 0xDA: _SSL_SESSION_OPT, Set options for a SSL session
- 0xDB: _SSL_CLI_CA_ADD, Add a client CA certificate
- 0xDC: _SSL_CLI_START, Start the SSL client
- 0xDD: _SSL_SEND_FLUSH, Queued user data will be flushed
- 0xDE: _SSL_CONN_CLOSE, Close a SSL connection

## Crypto functions

- 0xDF: _CRYPT_HASH, Build MD5/SHA1 checksum
- 0xE0: _CRYPT_CIPHER, Use symmetric crypto functions

## Others

- 0x09: _API_SLEEP, Sleep
- 0x0A: _API_MALLOC, Allocate memory
- 0x0B: _API_FREE, Free an allocated buffer
- 0x60: _API_SNMP_GET, Get internal TCP/IP SNMP variables
- 0x65: _API_FTP_GET_LOGIN, Get FTP server login counters
- 0x66: _API_TELNET_GET_LOGIN, Get Telnet server login counters
- 0x67: _API_GET_TELNET_STATE, Test Telnet session active
- 0x70: _GET_INSTALLED_SERVERS and interfaces
- 0x71: _RECONFIG_ETHERNET, Reconfigure Ethernet interface
- 0x74: _TCPIP_STATISTICS, Access packet counters

- 0x78:_GETMEMORY_INFO, Report TCP/IP memory usage
- 0x79:_SET_SERVER_IDLE_TIMEOUT, Control FTP/Telnet timeout
- 0x8D:_GET_IFACE_ENTRIES, Read table of TCP/IP device interfaces
- 0x8E:_SET_IFACE_MTU, Set MTU of a device interface
- 0x8F:_GET_IFACE_ENTRIES_EXT, Read table of TCP/IP device interfaces
- 0xC0:_GET_IPV6_IFACE_ENTRIES, Read IPv6 information for each interface

---

## Interrupt 0xAC service 0x00: Install dynamic link, then call function

This function installs a far JMP opcode in the vicinity of the return instruction pointer to enter the specified USB API function directly from the user code. This function is provided for efficiency in real-time applications.

## Parameters

*AH*
0x00

*AL*
Number of TCP/IP API function that is to be linked

## Return Value

Instruction Pointer is moved back 10 bytes so that newly installed far JMP will execute immediately.

(Refer to respective linked function's documention for further return value information.)

Note: All dynamically linked functions are free to use all registers except for DS, SI, DI, and BP registers which are preserved. This matches Paradigm C, Borland C++ 5.02 and Microsoft Visual C 1.52 calling conventions.

## Comments

Dynamic linking is only implemented for a few TCP/IP API functions!

A JMP FAR instruction will be written 10 bytes before the user's return instruction pointer. This far jump will transfer control directly into the RTOS from the user code when executed. This new far JMP will execute immediately upon return from this software interrupt due to the instruction pointer adjustment performed within the call.

Interrupts are masked and restored to their state on entry.

Since the code at calling site will be patched by this RTOS call, it is important that it has exactly the following form (the link index carried in AL register will vary)

```
JMP SHORT over_patch  ; Occupies 2 bytes of instruction code here
NOP    ; Never executed (1 byte)
NOP    ; Never executed (1 byte)
NOP    ; never executed (1 byte)
over_patch:  ; Linker overwrites above 5 bytes with JMP FAR instruction
MOV AX,000A2h  ; AH = 0x00, AL = 0x90 = index for Dev_Recv_Interface()
INT 0ACh
```

## Developer Notes

This API service is intended to be used by the RTOS C-Library exclusively.
It should not be used within user applications.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x01: API_OPENSOCKET, open a socket

Creates an IPv4 end-point for communication and returns a socket descriptor (i.e. a handle).

## Parameters

*AH*

0x01 (= API_OPENSOCKET)

*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

## Comments

This function provides the BSD socket() functionality.

## Related Topics

Open IPv6 socket **API_OPENSOCKET_IPV6**
Close socket **API_CLOSESOCKET**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x02: API_CLOSESOCKET, 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**
Open IPv4 socket **API_OPENSOCKET**
Open IPv6 socket **API_OPENSOCKET_IPV6**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x03: API_BIND, bind TCP or UDP server socket

Bind a unnamed socket with an IP address and port number.

## Parameters

*AH*
0x03 (= API_BIND)

*BX*
Socket descriptor

## SC1x Parameters

*DX:SI*
Pointer to a `sockaddr_in` **structure**, the first three elements of which must be filled in by caller prior to calling.

## SC1x3/SC23 Parameters

*DX:SI*
Pointer to a **sockaddr_in** structure (for IPv4 sockets) or to a **sockaddr_in6** structure (for IPv6 sockets). Caller must set the `sin_family, sin_port` and `sin_addr` (or `sin6_family, sin6_port` and `sin6_addr` for IPv6) elements prior to calling.

## Return Value

DX = 0 success AX: 0
DX != 0 AX: contains **error** code

## Comments

The bind call sets a specific port number as an application's source port number. Otherwise a random 16-bit source port number will be used when no bind call is made.

The bind call is necessary for server applications in order that clients can connect to them at the agreed upon ("well known") port number.

The older TCP and UDP echo client examples also used 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 number.

## Related Topics

Get IP address of the **Ethernet** interface
Convert ASCII IP address to **binary**
For proper port number byte order: **htons** function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x04: API_CONNECT, Connect to another socket

For 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

## SC1x Parameters

*DX:SI*
Pointer to a **sockaddr_in** structure containing host's IP address and port number.

## SC1x3/SC23 Parameters

*DX:SI*
Pointer to a **sockaddr_in** (for IPv4 sockets) or **sockaddr_in6** structure (for IPv6 sockets) containing

host's IP address and port number.

## Return Value

DX = 0 success AX: 0
DX != 0 AX: contains **error** code

## SC1x Comments

The caller must fill in the first three elements (`sin_family, sin_port` and `sin_addr`) of the **sockaddr_in** data structure at [DX:SI] prior to calling here.

## SC1x3/SC23 Comments

The caller must fill in the `sin_family, sin_port` and `sin_addr` elements of the **sockaddr_in** structure (IPv4) or the `sin6_family, sin6_port` and `sin6_addr` elements of the **sockaddr_in6** structure (IPv6) at [DX:SI] prior to calling here.

## Related Topics

Open IPv4 socket - **API_OPENSOCKET**
Open IPv6 socket - **API_OPENSOCKET_IPV6**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x05: API_RECVFROM, Receive UDP message

For UDP only, receives message from another socket.

## Parameters

*AH*
0x05 (= API_RECVFROM)

*BX*
Socket descriptor

*DX:SI*
Pointer to a **recv_params** data structure which caller must fill in prior to call.

## Return Value

DX = 0: Success, AX number of received bytes where 0 bytes indicates a timeout
DX != 0: Failure, AX contains **error** code

## Comments

This function will output the received data to the buffer referenced by `recv_params.bufferPtr` pointer. Up to `recv_params.bufferLength` bytes will be accepted. The AX return value indicates the number of bytes put into the buffer.
API_RECVFROM can only return complete UDP datagrams at the user provided buffer. If the next waiting datagram at the internal socket receive queue has size larger than the specified `recv_param.bufferlength`, API_RECVFROM returns error code 240 ("Message too long") without writing the datagram into the user buffer. You can use **API_GETRCV_BYTES** to detect the size of the next waiting datagram at the internal receive queue of this socket.

The TCP/IP stack's timing resolution is 100 milliseconds. Therefore if a timeout value is set in the **recv_params** structure of less than 100 milliseconds, the RECV API call will timeout after 100 milliseconds.

## SC1x Comments

The sender can be identified by the IP address and port number reported in the **sockaddr_in** data structure.

## SC1x3/SC23 Comments

The sender can be identified by the IP address and port number reported in the **sockaddr_in** structure (for IPv4 sockets) or in the **sockaddr_in6** structure (for IPv6 sockets).

## Related Topics

**API_SENDTO** - Send UDP datagram
**API_RECV** - Receive TCP data
**API_SELECT** - Socket select() operation
**API_GETRCV_BYTES** - Get waiting bytes on a socket
**API_OPENSOCKET_IPV6** - Open IPv6 socket

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x06: API_SENDTO, Transmit a UDP datagram

For UDP only, transmits message to another transport end-point.

## Parameters

*AH*

0x06 (= API_SENDTO)

*BX*

Socket descriptor

*DX:SI*
Pointer to a **send_params** data structure which caller must fill in prior to call.

## Return Value

DX = 0: Success, AX contains the number of bytes sent
DX != 0: Failure, AX contains **error** code

## Comments

On success, the return value in AX register indicates the actual number of bytes sent.

## SC1x Comments

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.

## SC1x3/SC23 Comments

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 (for IPv4 sockets) or by the **sockaddr_in6** structure (for IPv6 sockets) referenced by the `send_params.toPtr` pointer.

## Related Topics

**API_RECVFROM** - Receive UDP datagram
**API_SEND** - Send TCP data

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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**.

## Parameters

*AH*
0x07 (= API_HTONS)

*BX*
short value

## Return Value

DX = 0, AX contains converted value

## Comments

This is used to convert port numbers; e.g. htons(7).

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x08: API_INETADDR, Convert an IP-String to binary

Converts a dotted decimal IP address string to an unsigned long in **network byte order**.

## Parameters

*AH*
0x08 (= API_INETADDR)

*BX:SI*
Pointer to the IPv4 dotted decimal string set by caller. This ASCII string must be zero terminated.

*ES:DI*
Pointer to a 32 bit unsigned long variable, where this function outputs the converted IPv4 address value.

## Return Value

DX = 0 AX = 0, [ES:DI] contains converted 32 bit binary value
DX != 0 AX = -1, syntax error

## Related Topics

Get IPv4 address of the **Ethernet** interface
Convert binary IPv4 address to **ASCII** dotted decimal
Convert IPv4 or **IPv6** address ASCII string to binary

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x09: API_SLEEP, Sleep

The application sleeps for a specified number of milliseconds.

## Parameters

*AH*
0x09 (= API_SLEEP)

*BX*
milliseconds

## Return Value

DX = 0, AX = 0

## Comments

This call has been superceded by the similar RTOS API call **RTX_SLEEP_TIME**. This call here is maintained in the TCP/IP API for compatibility with earlier @CHIP-RTOS version.

For new applications, the RTX function with more meaningful return values is recommended.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x0A: API_MALLOC, Allocate memory

This function allocates memory direct from the @CHIP-RTOS heap.

## Parameters

*AH*
0x0A (= API_MALLOC)

*BX*
size in bytes

## Return Value

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

## Comments

This function has been replaced by the DOS **Int21h 0x48** API and is maintained here in the TCP/IP API only for compatibility with earlier @CHIP-RTOS versions.

For new applications, the DOS interrupt 0x21 function is recommended.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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 , AX = 0 success
DX != 0 , AX = 0 free failed

## Comments

This out dated function releases memory allocated with **API_MALLOC**. It is maintained here only for compatibility with earlier @CHIP-RTOS versions.

For new applications, the DOS interrupt 0x21 memory allocate **service 0x48** and release **service 0x49** are recommended.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

## Comments

If the specified socket is an UDP socket, API_GETRCV_BYTES will return the size (in bytes) of the next waiting datagram at the internal socket receive queue.

## Related Topics

**API_RECVFROM** - UDP Receive
**API_RECV** - TCP Receive

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x0D: API_ACCEPT, Accept the next incoming connection

This API extracts the first connection on the queue of pending connections (from **API_LISTEN**) and creates a new socket for this connection.

## Parameters

*AH*
0x0D (= API_ACCEPT)

*BX*
Socket descriptor (must be switched into listen mode using **API_LISTEN**)

## SC1x Parameters

*DX:SI*
Output Parameter: Pointer to a **sockaddr_in** structure (see tcpipapi.h)

## SC1x3/SC23 Parameters

*DX:SI*
Output Parameter: Pointer to a **sockaddr_in** structure (for IPv4 sockets) or to a **sockaddr_in6** structure (for IPv6 sockets).

## Return Value

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

## SC1x Comments

This call is used by a TCP server.

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

The new socket will have the same socket options as the listening socket (BX).

## SC1x3/SC23 Comments

This call is used by a TCP server.

On success, this function fills in the **sockaddr_in** structure or **sockaddr_in6** structure at [DX:SI] with the IP address and port number of the accepted connection.

The new socket will have the same socket options as the listening socket (BX).

## Related Topics

**API_SETSOCKOPT** - Set socket options
**API_OPENSOCKET_IPV6** - Open IPv6 socket
**IPV6** - Socket interface

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x0E: API_LISTEN, Listen for incoming connections

Places the socket into passive mode and sets the number of incoming TCP connections that the system will queue.

## 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

## Comments

This call is used by a TCP server.

## Related Topics

**API_ACCEPT** - Accept the next incoming connection

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x0F: API_SEND, Transmit TCP message

For TCP only, this API transmits a message to another transport end-point.

## Parameters

*AH*
0x0F (= API_SEND)

*BX*
Socket descriptor

*DX:SI*
Pointer to a **send_params** data structure which must be filled in by caller.

## Return Value

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

## Comments

API_SEND may be used only if the socket is in a connected state.

On success, the return value in AX contains the number of bytes which were successfully inserted into the socket's send queue.

If `flag` member of `send_params` is set to MSG_DONTWAIT the send call returns immediately. As much data as fits into the internal TCP buffer is accepted for transmission and the transmit byte count is returned in AX. If none of the data fits then -1 is returned in DX and error code 235 in AX.

If `flag` is set to MSG_BLOCKING, the send call blocks until enough internal buffer space is available or a socket error occurs. By default the blocking mode is set for all sockets at the open call. If a socket has been set to non-blocking with the **API_SETBLOCKINGMODE** function, the MSG_BLOCKING flag is ignored and the call will be non-blocking.

## Related Topics

**API_RECV** - Receive TCP data
**API_SENDTO** - Send UDP datagram

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x10: API_RECV, Receive TCP message

For TCP only, this API receives a message from another socket.

## Parameters

*AH*
0x10 (= API_RECV)

*BX*
Socket Descriptor

*DX:SI*
Pointer to **recv_params** data structure which must be filled in by user.

## Return Value

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

## Comments

API_RECV may only be used when the socket is in a connected state.

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`. The length of the message returned may be less than the parameter `bufferlength` of the **recv_param** structure. This is not an error.

If `flag` member of `recv_params` structure is set to MSG_DONTWAIT, the API_RECV call returns immediately. If no data is available, -1 is returned in DX with error code 235 in AX. If `flag` is set to MSG_BLOCKING, the API_RECV call waits for a message to arrive.

By default the blocking mode is set for all sockets at the open call. If a socket was set to non-blocking with the **API_SETBLOCKINGMODE** function, the MSG_BLOCKING flag is ignored and the call will be non-blocking.

The TCP/IP stack's timing resolution is 100 milliseconds. Therefore if a time-out value is set in **recv_params** data structure less than 100 milliseconds, the RECV API call will time-out after 100 milliseconds.

## Related Topics

**API_SEND** - Send TCP data
**API_RECVFROM** - Receive UDP datagram
**API_SELECT** - Socket select() operation
**API_GETRCV_BYTES** - Get waiting bytes on a socket

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x11: API_INETTOASCII, Convert IP address to IP string

Converts an unsigned long IPv4 address to an ASCII dotted decimal IP string.

## Parameters

*AH*
0x11 (= API_INETTOASCII)

*BX:SI*
Input Parameter: Pointer to the 32 bit IPv4 address in **network byte order**

*ES:DI*
Output Parameter: Pointer to the string buffer, where this function outputs the converted ASCII dotted decimal IP string. This buffer must have space for 16 bytes!

## Return Value

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

## Comments

The dotted decimal IP address ASCII string output to [ES:DI] buffer is zero terminated.

## Related Topics

Convert IPv4 address string to **binary**
Convert binary IPv4 or **IPv6** address to ASCII string

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0x12: API_RESETCONNECTION, Abort a connection on a socket

Applies only to TCP sockets.

### Parameters

*AH*

0x12 (= API_RESETCONNECTION)

*BX*

Socket descriptor

### Return Value

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

### Comments

This API does not close the socket. (The **API_CLOSESOCKET** API must be used to close the socket.)

The reset socket cannot be used for a new connection. You must close the socket and open a new one to get a new connection.

### Related Topics

Close socket **API_CLOSESOCKET**

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

## Interrupt 0xAC service 0x13: API_SETLINGER, Set linger time on close

Applies only to TCP sockets.

### Parameters

*AH*

0x13 (= API_SETLINGER)

*BX*

Socket descriptor

*CX*

Linger time in seconds, default: 20 seconds, 0 means linger turned off.

## Return Value

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x14: API_SETREUSE, Set reuse option on a listening socket

Sets **SO_REUSEADDR** socket option to '1'.

## Parameters

*AH*
0x14 (= API_SETREUSE)

*BX*
Socket descriptor

## Return Value

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

## Comments

This API applies only to TCP sockets. The reuse option is necessary if a listening socket is closed and then a new socket is opened and **bound** to the same port as the earlier socket.

## Developer Notes

Since SC12 @CHIP-RTOS version 071, the **API_OPENSOCKET** call sets the SO_REUSEADDR option to '1' as default for all TCP sockets. So calling this API is no longer necessary.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x15: API_SETIPTOS, Set IP Type-Of-Service

Sets socket's default *Type-Of-Service* placed in 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

The 8 bits in the IP datagram *Type-Of-Service* field are defined as follows:

Bits 0: unused bit (must be 0)
Bits 1-4: Type of Service, see TCP/IP documentation e.g.
Section 3.2 of "*TCP/IP Illustrated, Volume 1*" by W. Richard Stevens.
At most one of these four bits is set.
Bits 5-7: "Precedence field" (unused today)

Note: Many routers ignore this IP datagram header field.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x16: API_SETSOCKOPT, Set options on socket

Controls option settings for specfied socket.

## Parameters

*AH*
0x16 (= API_SETSOCKOPT)

*BX*
Socket descriptor

*ES:DI*
Pointer to a **SetSocketOption** type data structure that specifies the socket options

## 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` data structure.

The socket options of an incoming connection (using **accept**) will be the same as the socket options from its listening socket.

See **SetSocketOption** type definition for example usage of this API function.

## Related Topics

**API_GETSOCKOPT** - Get socket options

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x17: API_GETSOCKOPT, Get options on socket

Reads out an option setting for a specified socket.

## Parameters

*AH*
0x17 (= API_GETSOCKOPT)

*BX*
Socket descriptor

*ES:DI*
Pointer to **GetSocketOption** data structure.

## Return Value

DX = 0 success, Buffer pointed to by the `optionValue` member of the `GetSocketOption` structure 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 `optionLength` member is both an input and an output parameter. The length of the buffer must be specified at the location referenced by `optionLength` and this length must be sufficient for the size of the requested option data. On success, this API writes to the `optionLength` location the number of bytes output to the `optionValue` buffer.

The socket options of an incoming connection (using **accept**) will be the same as the socket options from its listening socket.

## Related Topics

**GetSocketOption** typedef with option names
**API_SETSOCKOPT** - Set socket options

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x18: API_SETBLOCKINGMODE, Set socket mode

Sets 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 will instead return immediately.

Example usage of non-blocking mode:

The **connect**call returns for a non-blocking socket with -1 in DX and error code 236 or 237 if the connection was not completed. The user can call connect within a loop, periodically polling for connection completion condition while performing some other activity, until a successful connection is established. In this case the connect function returns error code 0 or 256 (socket is already connected).

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x19: API_REGISTERCALLBACK, Register a user callback function

Register a user callback function for events occurring on a TCP socket.

## Parameters

*AH*
0x19 (= API_REGISTERCALLBACK)

*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 TCP/IP stack of the IPC@CHIP® is able to execute a registered user callback function, if one or more from specified events happens on a TCP socket connection.

Note that most of the call back calls will be made in the context of the drivers receive task (e.g. task ETH0). Therefore processing should be kept at a minimum in the call back function. The user call back function should set a flag or increase a counter and signal the user application task.

The callback function must be of the following type (Borland C):

```
void huge socketCallBackFunc(int socketdescriptor,
                             int eventFlags)
```

A socket callback may be removed by calling this API function with both a null null pointer in ES:DI and value 0 in register CX. **Closing** the socket also deactivates the callback. Before exiting a program, any sockets which have callbacks installed must either be closed or have the callback removed. Otherwise a callback could be present which references a program no longer loaded into memory.

The input parameter `eventFlags` to the callback is a bit field indicating the event(s) that have occurred.

The set of events for either the CX event mask argument to this API or the `eventFlags` callback

parameter are defined in TCPIPAPI.H as follows:

```
#define CB_CONNECT_COMPLT   0x0001   // Connection complete
#define CB_ACCEPT           0x0002   // Remote client has established
                                     //  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 occurred on the connection
#define CB_RESET            0x0080   // Peer has sent a rest on the connection
#define CB_CLOSE_COMPLT     0x0100   // close has been completed
```

## Related Topics

**API_REGISTERCALLBACK_PASCAL** For Pascal TCP callback functions
**IP_USER_CB** Install IP callback function
**ARP_USER_CB** Install ARP callback function
**IP_USER_CB_SND** Install IP callback function on outgoing packets

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x1A: API_SELECT, Wait on socket events

Wait on specified events for a selected set of sockets.

## Parameters

*AH*

0x1A (= API_SELECT)

*SS:BX*

Pointer to the first parameter passed to either the **select()** or **select_ms()** C-library API. In either case this first parameter is the `nfds` argument. The other parameters follow this one on the program stack, per the _cdecl calling convention.

## Return Value

DX = 0
AX = hit count, which is the number of sockets for which bits have been set in the **fd_set** output bit fields. On time-out or illegal input parameters, this count is zero.

## Comments

This API provides functionality similar to the Berkeley sockets standard **select()** function. The calling thread can wait for events on a set of sockets. The function will return after either a time-out period or when one of the specified events occurs on one of the selected sockets.

Socket selection is specified by the input **fd_set** data structures. Refer to the C-library function

documentation for more information reqarding the set of input parameters that appear here at [SS:BX].

## Beck C-Library Usage

**select()**
**select_ms()**

## Related Topics

**API_RECV** Receive TCP message
**API_RECVFROM** Receive UDP message

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x20: API_REGISTERCALLBACK_PASCAL, Pascal user callback

Registers a user callback function written in Pascal for events occurring on a TCP socket.

## 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 number of Pascal TCP callback functions is limited to 10.

The first action required in your callback function is to read the pointer to the **PacCallBack** record passed in registers ES:DI. This can be accomplished by implementing the callback function as follows (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. This is done by calling this API function with a null pointer in ES:DI and value 0 in register CX.

The event **flags** are defined as follows:

```
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;
```

## Related Topics

**API_REGISTERCALLBACK** For C language callback functions

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | n/a | n/a |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x21: API_GET_SOCKET_ERROR, Get last socket error.

Returns the last error which occurred on the specified socket.

## Parameters

*AH*

0x21 (= API_GET_SOCKET_ERROR)

*BX*

Socket descriptor

## Return Value

DX =0: success AX: contains last socket **error** code

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x22: API_GET_TCP_STATE, Find TCP socket and return state.

Search for a TCP socket at a given local port number, returning the socket state, IP address and port number of the remote peer (if any).

## Parameters

*AH*

0x22 (= API_GET_TCP_STATE)

*BX*

Local port (e.g. htons(23) for telnet)

*ES:DI*

Pointer to storage 32 Bit remote IP address

## Return Value

AL: contains last TCP socket state (see below)
If AL!=20 and AL greater or equal to 2 (a TCP connection is established), the storage at ES:DI holds the 32 bit IP address of the connected remote peer. CX contains remote peer port number.

## Comments

Possible TCP socket states:

0: CLOSED
1: LISTEN
2: SYN_SENT
3: SYN_RECEIVED
4: ESTABLISHED
5: CLOSE_WAIT
6: FIN_WAIT_1
7: CLOSING
8: LAST_ACK
9: FIN_WAIT_2
10: TIME_WAIT
20: INVALID

## SC1x3/SC23 Comments

Deprecated, because only IPv4 addresses can be stored at the provided 32-bit buffer [ES:DI].
Use **API_GET_TCP_STATE_IP_ALL** instead.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.04 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x23: API_FINDALL_SOCKETS, Get information about all open sockets.

Find/return information about all open sockets at the internal socket table.

## Parameters

*AH*

0x23 (= API_FINDALL_SOCKETS)

*ES:DI*

Output Parameter: Pointer to an array of CX **SocketInfo** data structure in the user's memory space into which this API will report the socket information.

*CX*

Size of the array (maximum number of table entries reported).

## Return Value

AX holds the number of all open sockets. The user provided array at [ES:DI] holds the socket information for up to CX sockets (the lesser of two register counts: AX, CX).

## Comments

If CX >= AX then all sockets in the socket table have been reported.

Note that the `localPort` and `remotePort` members of **SocketInfo** data structures are in little endian format. The `IfIpAddress` and `remoteIP` members remain in network byte order (big endian).

Below is an example of using API_FINDALL_SOCKETS with the `FindAllOpenSockets` C library wrapper function. This example lists up to 64 current entries in the TCP/IP socket table:

## SC1x3/SC23 Comments

Deprecated, due to that only IPv4 addresses can be stored in the **SocketInfo** data structure. The **API_FINDALL_SOCKETS_IP_ALL** function should instead be used, which also covers IPv6.

## SC1x Comments

This function is only available in @CHIP-RTOS versions which contain the SNMP MIB feature.

```
SocketInfo s[64];
int avail, i;

avail = FindAllOpenSockets(s, 64);
if (avail > 64) avail = 64;

for (i = 0; i < avail; i++)
{
   printf("\r\nSocket index %u Protocol %u, LocalPort %u",
     s[i].socIndex, s[i].protocol, s[i].localPort);
}
```

## Related Topics

**SocketInfo** data structure
**API_FINDALL_SOCKETS_IP_ALL** Get information about all open sockets

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x24: API_GET_TCP_STATE_IP_ALL, Find TCP sockets and return state

Search for a TCP socket at a given local port number, returning the socket state, IP address and port number of the remote peer (if any).

## Parameters

*AH*
0x24 (= API_GET_TCP_STATE_IP_ALL)

*BX*
Local port (e.g. htons(23) for telnet)

*ES:DI*
Pointer to storage 16 Byte (IPv6 or IPv4) remote IP address

## Return Value

AL: contains last TCP socket state (see below)
If AL!=20 and AL greater or equal to 2 (a TCP connection is established), the storage at ES:DI holds the 128 bit IPv6 (or mapped IPv4) address of the connected remote peer. CX contains the remote peer's port number. You can use the provided C-library macro IN6_IS_ADDR_V4MAPPED to determine if the stored address is a mapped IPv4 address.

## Comments

Possible TCP socket states:

0: CLOSED

1: LISTEN
2: SYN_SENT
3: SYN_RECEIVED
4: ESTABLISHED
5: CLOSE_WAIT
6: FIN_WAIT_1
7: CLOSING
8: LAST_ACK
9: FIN_WAIT_2
10: TIME_WAIT
20: INVALID

The API function **INET_NTOP** provides the address conversion to ASCII text.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x25: API_FINDALL_SOCKETS_IP_ALL, Get information about all open sockets.

Find/return information about all open sockets at the internal socket table.

## Parameters

*AH*

0x25 (= API_FINDALL_SOCKETS_IP_ALL)

*ES:DI*

Output Parameter: Pointer to an array of CX **SocketInfoIPv4v6** data structures in the user's memory space into which this API will report the socket information.

*CX*

Size of the array (maximum number of table entries reported).

## Return Value

AX holds the number of all open sockets. The user provided array at [ES:DI] holds the socket information for up to CX sockets (the lesser of two register counts: AX, CX).

## Comments

If CX >= AX then all sockets in the socket table have been reported.

Note that the `localPort` and `remotePort` members of **SocketInfoIPv4v6** data structures are in little endian format. The `IfIpAddress` and `remoteIP` members remain in network byte order (big endian). IPv4 addresses will appear in the `ip6Addr.ip6U32[3]` member of the **in6_addr** data structures.

Below is an example of using API_FINDALL_SOCKETS_IP_ALL with the `FindAllOpenSockets_IP_All` C-library wrapper function. This example lists up to 64 current entries in the TCP/IP socket table:

```
SocketInfoIPv4v6 s[64];
int avail, i;

avail = FindAllOpenSockets_IP_All(s, 64);
if (avail > 64) avail = 64;

for (i = 0; i < avail; i++)
{
   printf("\r\nSocket index %u Protocol %u, LocalPort %u",
     s[i].socIndex, s[i].protocol, s[i].localPort);
}
```

## Related Topics

**API_FINDALL_SOCKETS** - Find all IPv4 sockets

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x26: API_SHUTDOWN, Shutdown socket.

Shutdown a socket in read, write or both directions.

## Parameters

*AH*
0x26 (= API_SHUTDOWN)

*AL*
Direction: 0=Read, 1=write, 2=both

*BX*
socket descriptor

## Return Value

DX = 0: Success
DX != 0: Failure AX contains **error** code

## Comments

This API function only works on TCP sockets.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x27: API_RECVFROM_TO, Receive UDP message

For UDP only, Same as API_RECVFROM but returns also the destination address to which this packet was sent.

## Parameters

*AH*
0x27 (= API_RECVFROM_TO)

*BX*
Socket descriptor

*DX:SI*
Pointer to a **recv_params** data structure which caller must fill in prior to call.

*ES:DI*
Pointer to a user provided storage from type `sockaddr_in` **structure**, which contains (after sucessful return) the destination address of the received datagram.

## Return Value

DX = 0: Success, AX number of received bytes where 0 bytes indicates a timeout
DX != 0: Failure, AX contains **error** code

DX = 0: Success, AX number of received bytes where 0 bytes indicates a timeout
Storage pointed by ES:DI contains the destination address. DX != 0: Failure, AX contains **error** code

## Related Topics

**API_RECVFROM** - Receive UDP datagram

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x28: API_SENDTO_IFACE, Transmit a UDP datagram via specified interface

For UDP only, transmits message to another transport end-point via a specified interface.

## Parameters

*AH*
0x28 (= API_SENDTO_IFACE)

*BX*
Socket descriptor

*DX:SI*
Pointer to a **send_params** data structure which caller must fill in prior to call.

*ES:DI*
Driver handle pointer **Device handle**, which caller must fill in prior to call. TCPIP send the packet via this specified Device handle.

## Return Value

DX = 0: Success, AX contains the number of bytes sent
DX != 0: Failure, AX contains **error** code

## Comments

This function is required, to send packets with a IP multicast destination address over a specified device. If packets with multicast address are send by using the existing **API_SENDTO** function, TCPIP will use as default the fixed SC1xx ethernet device. On success, the return value in AX register indicates the actual number of bytes sent.

## SC1x Comments

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.

## SC1x3/SC23 Comments

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 (for IPv4 sockets) or by the **sockaddr_in6** structure (for IPv6 sockets) referenced by the `send_params.toPtr` pointer.

## Related Topics

**API_SENDTO** - Send UDP data
**DEV_GET_HANDLE_BYNAME** - Get device handle pointer

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.21 | V1.21 | V1.21 | V1.10 | V1.00 |

**Top of list**
**Index page**

## Interrupt 0xAC service 0x30: DNS_GET_HOST_BY_NAME, Get a host's address(es) by domain name (IPv4)

This function can be used to find the IPv4 addresses of a remote host associated to its domain name via the Domain Name Service (DNS).

## Parameters

*AH*

DNS_GET_HOST_BY_NAME (0x30)

*ES:BX*

Domain name of the host (e.g. "www.beck-ipc.com")

*AL*

Maximum number of IP addresses that the function is to return

*CX:DX*

Pointer to a buffer where the function is to store the found IP addresses; The buffer must be large enough to hold the number of IP addresses stated in the AL register. Ideally this should point to an array of type `struct in_addr`.

## Return Value

Number of addresses actually stored in the given buffer or negative **error code** in AX

## Comments

It is recommended to use the CLIB API call **getHostByName** instead of the software interrupt.

## Related Topics

**DNS_GET_HOST_BY_NAME2**
**DNS_GET_HOST_BY_ADDR**
**DNS_GET_HOST_BY_ADDR2**
**DNS_GET_MAIL_HOSTS**
**DNS_SET_NAME_SERVER**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x31: DNS_GET_HOST_BY_NAME2, Get a host's address(es) by domain name (IPv6)

This function can be used to find the IPv6 addresses of a remote host associated to its domain name via the Domain Name Service (DNS).

## Parameters

*AH*
DNS_GET_HOST_BY_NAME2 (0x31)

*ES:BX*
Domain name of the host (e.g. "www.beck-ipc.com")

*AL*
Maximum number of IP addresses that the function is to return

*CX:DX*
Pointer to a buffer where the function is to store the found IP addresses; The buffer must be large enough to hold the number of IP addresses stated in the AL register. Ideally this should point to an array of type `struct in6_addr`.

## Return Value

Number of addresses actually stored in the given buffer or negative **error code** in AX

## Comments

It is recommended to use the CLIB API call **getHostByName2** instead of the software interrupt.

## Related Topics

**DNS_GET_HOST_BY_NAME**
**DNS_GET_HOST_BY_ADDR**
**DNS_GET_HOST_BY_ADDR2**
**DNS_GET_MAIL_HOSTS**
**DNS_SET_NAME_SERVER**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x32: DNS_GET_HOST_BY_ADDR, Get a host's domain name by address (IPv4)

This function can be used to find the (canonical) domain name associated with a host by one of its IPv4 addresses via the Domain Name Service (DNS).

## Parameters

*AH*
DNS_GET_HOST_BY_ADDR (0x32)

*ES:BX*

Pointer to the host's IP address of type `struct in_addr`

*CX:DX*
Pointer to a buffer where the function is to store the domain name; A domain name can be up to 254 characters long. Adding one byte for the trailing zero byte of the string, the buffer must have a size of at least 255 bytes.

## Return Value

Zero in case of success or negative **error code** in AX

## Comments

It is recommended to use the CLIB API call **getHostByAddr** instead of the software interrupt.

## Related Topics

**DNS_GET_HOST_BY_NAME**
**DNS_GET_HOST_BY_NAME2**
**DNS_GET_HOST_BY_ADDR2**
**DNS_GET_MAIL_HOSTS**
**DNS_SET_NAME_SERVER**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x33: DNS_GET_HOST_BY_ADDR2, Get a host's domain name by address (IPv6)

This function can be used to find the (canonical) domain name associated with a host by one of its IPv6 addresses via the Domain Name Service (DNS).

## Parameters

*AH*
DNS_GET_HOST_BY_ADDR2 (0x33)

*ES:BX*
Pointer to the host's IP address of type `struct in6_addr`

*CX:DX*
Pointer to a buffer where the function is to store the domain name; A domain name can be up to 254 characters long. Adding one byte for the trailing zero byte of the string, the buffer must have a size of at least 255 bytes.

## Return Value

Zero in case of success or negative **error code** in AX

## Comments

It is recommended to use the CLIB API call **getHostByAddr2** instead of the software interrupt.

## Related Topics

**DNS_GET_HOST_BY_NAME**
**DNS_GET_HOST_BY_NAME2**
**DNS_GET_HOST_BY_ADDR**
**DNS_GET_MAIL_HOSTS**
**DNS_SET_NAME_SERVER**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x34: DNS_GET_MAIL_HOSTS, Get mail hosts

Use this function to determine the mail host(s) of a domain, i.e. host accepting internet mail for the domain.

## Parameters

*AH*
DNS_GET_MAIL_HOSTS (0x34)

*ES:BX*
Domain name to find mail hosts for (e.g. "beck-ipc.com")

*AL*
Maximum number of host names that the function is to return

*CX:DX*
Pointer to an array of type `DnsMailHost` where domain name and preference value of the mail hosts will be stored; The array must be large enough to hold the number of entries passed in AL register.

## Return Value

Number of mail hosts actually stored or negative **error code** in AX

## Comments

It is recommended to use the CLIB API call **getMailHosts** instead of the software interrupt.

## Related Topics

DNS_GET_HOST_BY_NAME
DNS_GET_HOST_BY_NAME2
DNS_GET_HOST_BY_ADDR
DNS_GET_HOST_BY_ADDR2
DNS_SET_NAME_SERVER

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x35: DNS_SET_NAME_SERVER, Add prioritised name server

This function can be used to add a prioritised name server which will be queried before all other name servers. Normally the DNS resolver determines name server addresses automatically from network devices configured via DHCP or PPP. In addition you can specify two name servers in the CHIP.INI configuration file. Name servers found via DHCP have the lowest priotity. Next come name servers found via PPP, then the two configured in CHIP.INI. The name server added here has the highest priority. Note that calling this function a second time overwrites the name server address added first.

## Parameters

*AH*
DNS_SET_NAME_SERVER (0x35)

*CX*
Address family, AF_INET for IPv4 or AF_INET6 for IPv6

*ES:BX*
Pointer to the IP address of the name server; should point to a structure of either type `struct in_addr` or `struct in6_addr`.

## Return Value

Zero in case of success or negative error code in AX

## Comments

It is recommended to use the CLIB API call setNameServer instead of the software interrupt.

## Related Topics

DNS_GET_HOST_BY_NAME
DNS_GET_HOST_BY_NAME2
DNS_GET_HOST_BY_ADDR
DNS_GET_HOST_BY_ADDR2
DNS_GET_MAIL_HOSTS

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x40: PPPCLIENT_INSTALLED, Check if PPP client installed.

Tests if PPP client services are available in this @CHIP-RTOS version and not disabled via CHIP.INI configuration.

## Parameters

*AH*
0x40 (= PPPCLIENT_INSTALLED)

## Return Value

AX = 0: PPP client is not installed
AX != 0: PPP client is installed

## Related Topics

PPPCLIENT_GET_STATUS API - PPP client status

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x41: PPPCLIENT_OPEN, Open a PPP connection

Opens a PPP client 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 see the **PPPClient_Init** data structure documentation.

Note:  **Only one PPP client connection can be open at a time!!**

## SC1x3/SC23 Comments

This API applies to IPv4 PPP connections.

## Related Topics

**PPPCLIENT_OPEN_IPV6** - Open IPv6 PPP client connection
**PPPCLIENT_CLOSE** - Closes PPP client connection

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x46:  PPPCLIENT_OPEN_IPV6, Open a PPP connection

Opens a PPP client connection with IPv6 Control protocol (IPv6CP).

## Parameters

*AH*
0x46 (= PPPCLIENT_OPEN_IPV6)

*ES:DI*
Pointer to a **PPPClient_Init_IPv6** type data structure (declared in 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

IPv6 address configuration is done by a auto negotiation process.  At connection time the peers negotiate an Interface-ID for the PPP device and auto configure their link-local scope addresses.  After sucessful return of this function, the `PPPClieipAddrStr` and `PPPClieRemipAddrStr` members of the **PPPClient_Init_IPv6** data structure are filled with the negotiated IPv6 addresses.  By default they contain the **::** zero address.

Refer to the PPP6CLIE.C example for how to use this API.  Also see the **PPPClient_Init_IPv6** data

structure documentation.

Note: **Only one PPP client connection can be open at a time!!**

## Related Topics

**PPPCLIENT_OPEN** - Open IPv4 PPP client connection
**PPPCLIENT_CLOSE** - Closes PPP client connection

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x42: PPPCLIENT_CLOSE, Close a PPP client connection

Closes 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 close timed out

## Comments

After this API returns, the PPP client after closing the PPP session executes the modem hang-up commands specified in the **PPPClient_Init** data structure that was provided at the **PPPCLIENT_OPEN** call. Therefore we recommend waiting approximately 10 seconds before opening a subsequent PPP connection to assure that this hang-up command sequence has completed.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x43: PPPCLIENT_GET_STATUS, Get PPP client status

Gets PPP client's current status.

## Parameters

*AH*
0x43 (= PPPCLIENT_GET_STATUS)

## Return Value

AX = -2 (=API_NOT_SUPPORTED), DX =- 2: PPP client **task** is not running
AX = -1 (=API_ERROR), DX = PPP client **status**
AX >= 0, then AX = PPP client status, DX unchanged

## SC1x Comments

If the PPP client is not supported in this @CHIP-RTOS version, then the value returned in AX is zero and DX is set to -1 (=API_ERROR). In this case an error message is issued to the console.

## Related Topics

**PPPCLIENT_INSTALLED** API - PPP client installed check

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x44: PPPCLIENT_GET_DNSIP, Get DNS IP address

Returns the *Domain Name System* (DNS) addresses as negotiated by the remote PPP server.

## Parameters

*AH*
0x44 (= PPPCLIENT_GET_DNSIP)

*BX*
1: Get primary DNS address, 2: Get secondary address

*ES:DI*
Output Parameter: Pointer to an unsigned long variable where the requested DNS IPv4 address will be stored by this API.

## Return Value

AX = -2 (=API_NOT_SUPPORTED), DX = -2: PPP client is not **installed**.
AX = -1 (=API_ERROR) DX contains **error** code.
AX = 0, then unsigned long at [ES:DI] holds the DNS IP address.

## Comments

The IP address output to [ES:DI] is in **network byte order**.

## SC1x3/SC23 Comments

At the current @Chip-RTOS version, only IPv4 addresses are supported.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.04 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x45: PPPCLIENT_SET_OPTIONS, Set options for the PPP client

Sets the PPP options for the PPP client.

## Parameters

*AH*

0x45 (= PPPCLIENT_SET_OPTIONS)

*ES:DI*

Pointer to an array of **PPP_Option** data structures. This array is terminated by first `PPP_Option` structure with a null `optionValuePtr` member (see example below).

## Return Value

AX = -3 PPP Connection already established AX = -2 PPP client is not **installed**.
AX = -1 Invalid Option(s) AX = 0 Success

## Comments

If you want to use this function, you have to call it before opening a connection (see related links below). Setting options during an established connection will not work!

The settings are only valid for the next established connection. The data structures referenced here with ES:DI must persist until after the subsequent **PPP client open** function completes.

Here is an example which sets three PPP options using the Beck C-Library wrapper functions:

```
unsigned int  ipcp_comp   = 1;
unsigned long DNS_Pri_IP  = 0L;
unsigned long DNS_Sec_IP  = 0L;

PPP_Option My_Options[] = {

     // Allow remote peer to use VJ TCPIP header compression.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_ALLOW, PPP_IPCP_COMP_PROTOCOL,
     (const char *)&ipcp_comp, sizeof(ipcp_comp)},

    // Allow remote peer to set primary DNS IP.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_PRI,
     (const char *)&DNS_Pri_IP, sizeof(DNS_Pri_IP)},

    // Allow remote peer to set secondary DNS IP.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_SEC,
     (const char *)&DNS_Sec_IP, sizeof(DNS_Sec_IP)},

    // This last PPP_Option is used to terminate array.
  { 0, 0, 0, NULL, 0}
} ;

//***** Call the C-Library functions as follows ****

 // Install options with CLIB function
PPP_Client_SetOptions(&My_Options[0]);  // Point to first member of array

PPP_Client_Open(&pppclient);            // Open the connection
```

## Related Topics

**PPPCLIENT_OPEN** open a PPP connection
**PPPCLIENT_CLOSE** close a PP connection

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x50: PPPSERVER_INSTALLED, Check if PPP server installed

Tests if PPP server is available in this @CHIP-RTOS version and not disabled via CHIP.INI **configuration**.

## Parameters

*AH*
0x50 (= PPPSERVER_INSTALLED)

## Return Value

AX = 0: PPP server is not installed
AX != 0: PPP server is installed

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x51: PPPSERVER_SUSPEND, Suspend PPP server task

Suspends the PPP server task, "PPPS".

### 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 in BX depends on your timeout entries for the modem **commands** specified in CHIP.INI. If this call returns with -1 in AX and DX, the most likely reason for this is that the modem commands were not finished within the timeout period specified in BX.

### Related Topics

**PPPSERVER_ACTIVATE** API - Reactivate PPP server
PPPSERVER **HANGUPTIMEOUTx** - CHIP.INI timeout for wait on answer from modem

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x52: PPPSERVER_ACTIVATE, Activate PPP server

Re-activate **suspended** PPP server.

### 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

This API will not activate the PPP server disabled from start up by the CHIP.INI **configuration** entry.

On success, the PPP server is now able to serve a connection.

Note that the timeout value in BX depends on your timeout entries for the modem **commands** specified in CHIP.INI. If this call returns with -1 in AX and DX, the most likely reason for this is that the modem commands were not finished within the timeout period specified in BX.

## Related Topics

**PPPSERVER_SUSPEND** API - Suspend PPP server
PPPSERVER **HANGUPTIMEOUTx** - CHIP.INI timeout for wait on answer from modem

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x53: PPPSERVER_GET_STATUS, Get server state

Gets 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 listed below

## Comments

PPP server states:

-1 Error state, should not happen
1 Server disabled
2 Server enabled, waiting for connection

3 PPP connection is established
4 Server tries to hang up modem
5 Server tries to initialize modem

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x54: PPPSERVER_GET_CFG, Get PPP server configuration

Gets the current main configuration data of the PPP server.

## Parameters

*AH*

0x54 (= PPPSERVER_GET_CFG)

*ES:DI*

Output Parameter: Pointer to `PPP_IPCfg_Data` data **structure** where this function will report the configuration data.

## Return Value

DX != 0 AX != 0: PPP server is not installed
DX = 0 The user `PPP_IPCfg_Data` structure at [ES:DI] is filled with the PPP server configuration data

## SC1x3/SC23 Comments

Only PPP IPv4 connections are supported by this API.

## Related Topics

**PPPSERVER_GET_CFG_IPV6** - Get PPP IPv6 configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x55: PPPSERVER_SET_OPTIONS, Set options for PPP server

Sets the PPP options for the PPP server.

## Parameters

*AH*

0x55 (= PPPSERVER_SET_OPTIONS)

*ES:DI*

Pointer to an array of `PPP_Option` data **structures**. This array is terminated by first `PPP_Option` structure with a null `optionValuePtr` member (see example below).

## Return Value

AX = -3: PPP Connection already established
AX = -2: PPP server is not installed
AX = -1: Invalid Option(s)
AX = 0: Success

## Comments

It is only possible to set PPP options when the PPP server is **suspended**. PPP server must be at **status** 1. Setting options while the PPP server is active (state 2,3,4,5) has no effect. The installed options will be applied as the PPP server is **re-activated**. The data structures referenced by ES:DI must persists until that time.

If you want to reset the options, call this function with a null pointer in ES:DI.

Here is an example which sets three PPP options using the C-Library wrapper functions:

```
unsigned int  ipcp_comp   = 1;
unsigned long DNS_Pri_IP  = 0L;
unsigned long DNS_Sec_IP  = 0L;

PPP_Option My_Options[] = {

   // Allow remote peer to use VJ TCPIP header compression.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_ALLOW, PPP_IPCP_COMP_PROTOCOL,
     (const char *)&ipcp_comp, sizeof(ipcp_comp)},

   // Allow remote peer to set primary DNS IP.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_PRI,
     (const char *)&DNS_Pri_IP, sizeof(DNS_Pri_IP)},

   // Allow remote peer to set secondary DNS IP.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_SEC,
     (const char *)&DNS_Sec_IP, sizeof(DNS_Sec_IP)},

    // This last PPP_Option is used to terminate array.
  { 0, 0, 0, NULL, 0}
} ;

 //***** Call the C-Library functions as follows ****

 // Suspend the PPP server
PPP_Server_Suspend(20, &error);
 // Install options with CLIB function
PPP_Server_SetOptions(&My_Options[0]);  // Point to first member of array
  // Re-activate the server
PPP_Server_Activate(20, &error);
```

## Related Topics

**PPPSERVER_SUSPEND** suspend PPP server task
**PPPSERVER_ACTIVATE** activate PPP server

**PPPSERVER_GET_STATUS** Get status of the PPP server

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x56: PPPSERVER_GET_CFG_IPv6, Get PPP server configuration

Gets the current main configuration data of the PPP server.

## Parameters

*AH*

0x56 (= PPPSERVER_GET_CFG_IPV6)

*ES:DI*

Output Parameter: Pointer to **PPP_IPCfg_Data_IPv6** data structure where this function will report the configuration data.

## Return Value

DX != 0 AX != 0: PPP server is not installed
DX = 0: The caller's `PPP_IPCfg_Data_IPv6` structure at [ES:DI] is filled with the PPP server IPv6 configuration data

## Comments

Only PPP IPv6 connections are supported by this API. IPv6 address configuration is done by a auto negotiation process. At connection time the peers negotiate an Interface-ID for the PPP device and auto configure their link-local scope addresses. If a connection is established, the `IP` and `RemoteIP` members of the **PPP_IPCfg_Data_IPv6** data structure are filled with the negotiated addresses. By default, they contain the **"::"** zero address.

## Related Topics

**PPPSERVER_GET_CFG** - Get PPP IPv4 configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x60: API_SNMP_GET, Get internal TCP/IP SNMP variables

Accesses *Management Information Base* (MIB) data structures residing inside the @CHIP-RTOS TCP/IP stack.

## Parameters

*AH*

0x60 (= API_SNMP_GET)

*AL*

1: Get pointer to **IfMib** data structure (only for the internal Ethernet device interface)
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 (only of the internal Ethernet device interface)
7: Get array of far pointers to **IfMib** entries of all current open TCP/IP device interfaces. (Local Loopback, Ethernet, PPP server, PPP client)
8: Get far pointer to unsigned long sysuptime variable (10 Hz counter).

*DS:SI*

If parameter AL==7, DS:SI must contain the address of a buffer into which up to CX far pointers to **IpMib** data structures can be reported by this API. Otherwise these registers are not referenced.

*CX*

If parameter AL==7, CX specifies the number of far pointers (capacity) which the user buffer at [DS:SI] can hold. Otherwise it is not referenced.

## Return Value

DX != 0 AX != 0 : @CHIP-RTOS without internal SNMP MIB variables

DX = AX = 0: Success, return values depend on AL input value.

If input parameter AL was 7:
The user provided array (still addressed by DS:SI) contains the far pointers to the `IfMib`entries of the currently installed device interfaces. CX is set to indicate the total number of currently installed devices. If CX is less than it was on entry, then the reported list of pointers is not complete due to lack of user buffer space.

Otherwise (AL was 1..6, or 8):
ES:DI contains a pointer to the requested @CHIP-RTOS data structure

## Comments

If this API call is used with AL==1-6, or 8, the pointers are granting direct access to the internal TCP/IP stack counters. It's recommended to execute these calls only one time at the start of your application.

For AL=7, the API call returns the pointers to the interface variables for the currently installed device interfaces. To maintain an up to date array of pointers for all currently opened device interfaces, it is necessary to periodically request (e.g. every 10 seconds) the array of pointers.

## SC1x Comments

**Note: These structures are only available in @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to the internal TCP/IP SNMP variables. The SNMP MIB variables are not a part of our current official 6 SC1x @CHIP-RTOS variants. It is necessary to order directly a @CHIP-RTOS variant which includes this feature.**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x65: API_FTP_GET_LOGIN, Get FTP server login counters

Accesses the FTP server login counters.

## Parameters

*AH*
0x65 (= API_FTP_GET_LOGIN)

## Return Value

DX != 0 AX != 0 : @CHIP-RTOS doesn't support FTP server

DX = AX = 0: Success
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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x66: API_TELNET_GET_LOGIN, Get Telnet server login counters

Accesses the Telnet server login counters.

## Parameters

*AH*
0x66 (= API_TELNET_GET_LOGIN)

## Return Value

DX != 0 AX != 0 : @CHIP-RTOS doesn't support Telnet server

DX = AX = 0: Success
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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x67: API_GET_TELNET_STATE, Test Telnet session active

Check if the Telnet server is currently handling an active Telnet session.

## Parameters

*AH*

0x67 (= API_GET_TELNET_STATE)

## Return Value

DX = -1 AX = -1 : @CHIP-RTOS doesn't support Telnet server

DX = 0 AX = 1: Telnet session is active
DX = 0 AX = 0: No Telnet session

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

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 requested 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x71: RECONFIG_ETHERNET, Reconfigure Ethernet interface

Reconfigures 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). Error code 237 indicates that an Ethernet interface configuration was already in progress.

## Comments

A new IP configuration set with the prompt **commands** `ip`, `netmask` and `gateway` (or the corresponding @CHIP-RTOS **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 @CHIP-RTOS API interrupt 0xA0 services **0x02**, **0x04**, **0x06** before using this function.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x72: DHCP_USE, Enable/Disable DHCP usage

Set/Reset DHCP usage by the Ethernet interface.

## Parameters

*AH*
0x72 (= DHCP_USE)

*AL*
0: DHCP not used, 1:DHCP_USE

## Return Value

-- none --

## 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 @CHIP-RTOS API interrupt 0xA0 services **0x02**, **0x04**, **0x06**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x73: DHCP_STAT, Get DHCP status of the Ethernet interface

Gets the 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: 0 System is not configured (is in progress)
DX: 1 System is configured by a DHCP Server
DX: 2 System configure retry failed (or no retry started before)

If DX == 1 ES:DI points to a **UserEthDhcp_Entry** data structure which contains the received DHCP configuration data.

## Comments

The returned DHCP configuration data must be treated by the user as read-only information. Do not write to this data structure!

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x74: TCPIP_STATISTICS, Access packet counters

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

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

`Packet_Count` structure **type** definition

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x75: PING_OPEN, Open and start ICMP echo requests

This function opens and starts periodic *Internet Control Message Protocol* (ICMP) echo request (ping) to a specified remote host.

## Parameters

*AH*
0x75 (= PING_OPEN)

*ES:DI*
Pointer to user's `Ping` **structure**, four members of which must be set by caller prior to call.

## Return Value

DX: -1, AX contains **error** code, Ping open failed
else
DX: socket descriptor, AX: 0

## Comments

**Important:**
If structure member `count` (set by user) is non-zero, the PING_STATISTICS call closes the ping socket automatically if structure member `received` has reached the `count` value. If structure member `count` is zero, ping process runs until PING_CLOSE is called.

## Related Topics

**PING_CLOSE** API
**PING_STATISTICS** API
Console **PING** command

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 inside `Ping` **structure** from PING_OPEN call.

## Return Value

DX: -1, AX -1, Ping close failed, should not happen, only if socket descriptor is invalid
else
DX: 0, AX: 0: success

## Comments

The ping socket is closed here.

## Related Topics

**PING_OPEN** API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 = 0, AX = 0: Success
DX = -1, AX = -1: Failure

## Comments

Structure at [ES:DI] is filled with the ping statistics on success.

**Important:**
If structure member `count` (set by user) is non-zero, the PING_STATISTICS call closes the ping socket automatically if structure member `received` has reached the `count` value. If structure member `count` is zero, ping process runs until PING_CLOSE is called.

## Related Topics

**PING_OPEN** API
**PING_CLOSE** API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |
|---|---|---|---|---|

Top of list
Index page

---

## Interrupt 0xAC service 0x78: GETMEMORY_INFO, Report TCP/IP memory usage

This function reports the maximum available and the currently used memory of the TCP/IP stack.

## Parameters

*AH*

0x78 (= GETMEMORY_INFO)

*ES:DI*

Output Parameter: Pointer to user's unsigned long where this API will report the maximum available TCP/IP memory byte count.

*DS:SI*

Output Parameter: Pointer to user's unsigned long where this API will report the currently used TCP/IP memory byte count.

## Return Value

DX: 0, AX 0 (The described memory sizes are stored at [ES:DI] and [DS:SI])

## Comments

The maximum available memory for the TCP/IP stack can be configured in `chip.ini` (see **TCPIPMEM**).

The amount of memory required by the TCP/IP stack depends on the number of open sockets and the size and number of transported data packets. For memory blocks equal or smaller than 4096 bytes, the TCP/IP stack allocates memory from this pre-allocated block. Once allocated, the TCP/IP stack does not release this memory back to the system. It will be internally recycled for further usage.

Memory blocks larger than 4096 bytes are allocated directly from the @CHIP-RTOS memory. The TCP/IP stack releases these blocks back to the IPC@CHIP® memory management. If your application requires a lot of memory you should avoid sending and receiving frames larger than 2048 bytes. Larger packets should be split into some smaller ones prior to sending.

With **BIOS** interrupt 0xA0 it is possible to install a user error handler function which will be called if the memory limit is reached.

## SC1x Comments

The default size of this memory block is 90 kBytes in the IPC@CHIP® Large version and 98 kBytes in the @CHIP-RTOS version including PPP.

## SC1x3/SC23 Comments

The default size of this memory block is 140 kBytes.

## Related Topics

BIOSINT API **Install a user fatal error handler**
IPC@CHIP® **TCP/IP memory** `chip.ini` Configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x79: SET_SERVER_IDLE_TIMEOUT, Control FTP/Telnet timeout

Set/Get the Idle timeout for either the FTP or Telnet server.

## Parameters

*AH*
0x79 (= SET_SERVER_IDLE_TIMEOUT)

*AL*
0: Set Timeout, Non-zero: Get Timeout

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

*DX*
Timeout value in seconds (If AL = 0 for set)

## Return Value

Set idle timeout -
DX = AX = 0: Success
AX = DX = -1: Server not provided

Get idle timeout -
AX = 0: DX contains timeout value
AX = DX = -1: Server not provided

## Comments

If input parameter AL is zero, this API inserts the new timeout value in DX into the CHIP.INI. If the Telnet timeout is changed, a reboot is not necessary for the new value to take affect. If the FTP timeout is changed, a reboot is required, before the new time-out value takes affect.

If AL is non-zero, this API returns the existing timeout for the specified server.

## Related Topics

**FTP Timeout** in `chip.ini` Configuration
**Telnet Timeout** in `chip.ini` Configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x7A: IP_USER_CB, Install IP callback function

This function allows the application programmer to install an callback function on incoming IPv4 packets.

### Parameters

*AH*
0x7A (= IP_USER_CB)

*ES:DI*
Pointer to IP callback function (or set to null to remove a previously installed callback)

### Return Value

DX: 0, AX 0 success

### Comments

The application programmer can implement a function of the type (Borland C):

```
typedef int (huge *MyIpCallbackFuncPtr)(
                IpUserCallbackInfo far *ipInfo );
```

If a function of this type is installed by the user, the TCP/IP stack will call this function for any incoming IP packet. Inside of this function the user is able to check the given IP parameters (which are available through the **IpUserCallbackInfo** parameter) and decide whether the TCP/IP stack should process this packet or ignore it.

If the callback function returns -1 the incoming packet will be ignored by the TCP/IP stack.

To remove the callback function, this API call must be called with a null pointer in ES:DI. ***Do not forget to uninstall the callback, if your application exits!***

### SC1x3/SC23 Comments

**Important:** The callback will only execute on incoming IPv4 packets.

### Related Topics

**API_REGISTERCALLBACK** Install TCP callback function
**API_REGISTERCALLBACK_PASCAL** For Pascal TCP callback functions
**ARP_USER_CB** Install ARP callback function
**IP_USER_CB_SND** Install IP callback function on outgoing packets

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |
|---|---|---|---|---|

Top of list
Index page

---

## Interrupt 0xAC service 0x7B: ARP_USER_CB, Install ARP callback function

This function allows the application programmer to install an ARP callback function.

### Parameters

*AH*

0x7B (= ARP_USER_CB)

*ES:DI*

Pointer to ARP callback function (or set to null to remove a previously installed callback)

### Return Value

DX: 0, AX 0 Success

### Comments

The application programmer can implement a function from the type described below:

```
typedef int (huge *MyArpCallbackFuncPtr)(
                ArpUserCallbackInfo_t far *arpInfo );
```

If a function of this type is installed by the user, the TCP/IP stack will call this function for any incoming ARP packet. The input parameter to the callback function allows access to the ARP packet. Inside of this function the user is able to check the content of the incoming ARP packet and decide whether the TCP/IP stack should process this packet or ignore it. If the callback function returns -1, the incoming packet will be ignored by the TCP/IP stack. If the size of the returned structure **ArpUserCallbackInfo_t** is equal or bigger 12, the structure now contains a **devicehandle_ptr** to determine the source device of the incoming packet.

For structure of an ARP packet see **ArpHeader**.

To remove the callback function, this API call must be called with a null pointer in ES:DI. ***Do not forget to uninstall the callback, if your application exits!***

### Related Topics

**API_REGISTERCALLBACK** Install TCP callback function
**API_REGISTERCALLBACK_PASCAL** For Pascal TCP callback functions
**IP_USER_CB** Install IP callback function
**IP_USER_CB_SND** Install IP callback function on outgoing packets

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

## Interrupt 0xAC service 0x7C: IP_USER_CB_SND, Install IP callback function on outgoing packets

This function allows the application programmer to install an IP callback function on outgoing IP packets.

## Parameters

*AH*
0x7C (= IP_USER_CB_SND)

*ES:DI*
Pointer to IP callback function (or set to null to remove a previously installed callback)

## Return Value

DX: 0, AX 0 success

## Comments

The application programmer can implement a function of the type (Borland C):

```
typedef int (huge *MyIpCallbackFuncPtr)(
                IpUserCallbackInfo far *ipInfo );
```

If a function of this type is installed by the user, the TCP/IP stack will call this function for any outgoing IP packet. Inside of this function the user is able to check the given IP parameters (which are available through the **IpUserCallbackInfo** type at the `ipInfo` parameter) and decide whether the TCP/IP stack should process this packet or ignore it.

If the callback function returns -1 the TCP/IP stack doesn't send this packet.

To remove the callback function, this API call must be called with a null pointer in ES:DI. ***Do not forget to uninstall the callback, if your application exits!***

## SC1x3/SC23 Comments

**Important:** The callback will only execute on incoming IPv4 packets.

## Related Topics

**API_REGISTERCALLBACK** Install TCP callback function
**API_REGISTERCALLBACK_PASCAL** For Pascal TCP callback functions
**IP_USER_CB** Install IP callback function
**ARP_USER_CB** Install ARP callback function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

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*
Device entry of the gateway, 0: Ethernet, 1: PPP server, 2: PPP client, 3: User device driver

*ES:DI*
Pointer to an unsigned long containing the gateway IP in **network byte order**.

*DX:SI*
If BX == 3 DX:SI must contain **User device handle pointer**.

### 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.

If the PPP server or PPP client is specified in BX, the gateway is set to the remote peer IP address.

### Related Topics

**DEL_DEFAULT_GATEWAY** API function
**DEV_OPEN_IFACE** API function
IP **Gateway** `chip.ini` Configuration
PPP server **Gateway** `chip.ini` Configuration

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0x81: DEL_DEFAULT_GATEWAY, Delete the default gateway

This function is used to delete the default gateway.

### Parameters

*AH*

0x81 (= DEL_DEFAULT_GATEWAY)

*ES:DI*

Pointer to an unsigned long containing the gateway IP to be deleted in **network byte order**.

## 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
IP **Gateway** `chip.ini` Configuration
PPP server **Gateway** `chip.ini` Configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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*

Output Parameter: Pointer to user's unsigned long variable where the gateway IP will be written.

## 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0x83: ADD_STATIC_ROUTE, Add a route for an interface

This function is used to add a route for an interface. It allows packets for a different network to be routed by the interface.

### Parameters

*AH*
0x83 (= ADD_STATIC_ROUTE)

*BX*
Device entry of the gateway, 0: Ethernet, 1: PPP server, 2: PPP client, 3: User device driver

*DX:SI*
If BX = 3, DX:SI must contain **User device handle pointer**.

*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:

### SC1x3/SC23 Comments

**Important:** The current SC1x3/SC23 Chip-RTOS version supports only route definitions based on the IPv4 protocol.

### Related Topics

**DEL_STATIC_ROUTE** API function

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

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 from 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 function's call.

## SC1x3/SC23 Comments

**Important:** The current SC1x3/SC23 Chip-RTOS version supports only route definitions based on the IPv4 protocol.

## Related Topics

**ADD_STATIC_ROUTE** API function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x88: DEL_ARP_ENTRY_BY_PHYS, Delete by physical address

This function deletes the entry from the ARP table by the given physical address.

## Parameters

*AH*
0x88 (= DEL_ARP_ENTRY_BY_PHYS)

*ES:DI*
Pointer to physical address buffer (6 Byte)

## Return Value

DX:  0, AX 0 success
DX: -1, AX contains error code

## Related Topics

ADD_ARP_ENTRY  Add ARP entry
GET_ARPROUTE_CACHE  Get ARP cache

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x89:  ADD_ARP_ENTRY, Add an entry to the ARP table

This function adds an entry into the *Address Resolution Protocol* (ARP) table.

## Parameters

*AH*
0x89 (= ADD_ARP_ENTRY)

*BX:SI*
Pointer to IP address (unsigned long)

*ES:DI*
Pointer to physical address (6 Byte)

## Return Value

DX = 0, AX = 0:  Success
DX = -1: Failure, AX contains error code

## Related Topics

DEL_ARP_ENTRY_BY_PHYS  Delete ARP entry by physical address
GET_ARPROUTE_CACHE  Get ARP/ROUTE cache

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x8A: GET_ARPROUTE_CACHE, Read ARP/Route cache table

This function returns all valid entries in the internal ARP/Route cache table.

## Parameters

*AH*

0x89 (= GET_ARPROUTE_CACHE)

*BX:SI*

Output Parameter: Pointer to a user provided array of **ARP entries** of `ArpRouteCacheEntry` data **structures** for storing ARP/Route cache table.

## Return Value

DX = 0, AX = 0, the user provided array at [BX:SI] contains the current ARP/Route cache table.
CX holds the number of valid entries in table.

## Comments

The array must provide **ARP entries** entries. The API call returns only a copy of the current device interface variables. In order to always have the current table status, it is necessary to cyclically (e.g. every 10 seconds) request the table with this API.

## Related Topics

**DEL_ARP_ENTRY_BY_PHYS** Delete ARP entry by physical address
**ADD_ARP_ENTRY** Add ARP entry

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x8D: GET_IFACE_ENTRIES, Read table of TCP/IP device interfaces

This function returns information about all installed TCP/IP device interfaces.

## Parameters

*AH*

0x8D (= GET_IFACE_ENTRIES)

*BX:SI*

Pointer to a user provided array of `Iface_Entry` data **structures** for storing the information.

*CX*

Number of data structures in array at [BX:SI]. Up to this number of device interfaces will be reported.

## Return Value

DX = 0, the user provided array at [BX:SI] contains the current device interface information
AX = Number of entries reported in array at [BX:SI].

## SC1x3/SC23 Comments

This function return only the IPv4 configuration of the found TCP/IP device interfaces.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x8E: SET_IFACE_MTU, Set MTU of a device interface

This function sets the MTU (Maximum Transfer Unit) size of a device interface.

## Parameters

*AH*
0x8E (= SET_IFACE_MTU)

*BX*
Interface number
0:internal ethernet interface
1:internal ppp server
2:internal ppp client
3:user specific: interface handle at es:di

*CX*
MTU

*ES:DI*
if bx==3, ES:DI must contain the **User device handle pointer**.

## Return Value

DX: 0, AX 0 success
DX:-1, AX contains **error** code

## Related Topics

**DEV_OPEN_IFACE** API function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |
|---|---|---|---|---|

Top of list
Index page

---

## Interrupt 0xAC service 0x8F: GET_IFACE_ENTRIES_EXT, Read table of TCP/IP device interfaces

This function returns information about all installed TCP/IP device interfaces.

### Parameters

*AH*
0x8F (= GET_IFACE_ENTRIES_EXT)

*BX:SI*
Pointer to a user provided array of `Iface_Entry` data **structures** for storing the information.

*CX*
Number of data structures in array at [BX:SI]. Up to this number of device interfaces will be reported.

### Return Value

DX = 0, the user provided array at [BX:SI] contains the current device interface information
AX = Number of entries reported in array at [BX:SI].

### Comments

This function is different from **GET_IFACE_ENTRIES**. It stores at the user provided array from type **structures** all ip configurations (IP multihoming on one interface) of each interface found, not only the first one.

### SC1x3/SC23 Comments

This function return only the IPv4 configuration of the found TCP/IP device interfaces.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0x90: ADD_IGMP_MEMBERSHIP, Install an IP multicast address entry

Install an IP multicast address entry to become a member of an IP multicast group.

### Parameters

*AH*

0x90 (= ADD_IGMP_MEMBERSHIP)

*ES:DI*
Pointer to multicast IPv4 address of 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 **error** code, invalid class D address, or no free entry in IGMP table available

## Comments

IP Multicasting is the Internet abstraction of hardware multicasting. It allows transmission of IP datagrams to a group of hosts that form a single multicast group. Membership in a multicast group is dynamic. Hosts may join or leave the 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 binary 1110. IP multicast addresses range from 224.0.0.0 through 239.255.255.255.

For the usage of IP multicasting on 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. In this mode it receives any incoming IP packet with the mapped Ethernet multicast address and forwards it to the TCP/IP layer. Each IP multicast packet will be sent with the mapped Ethernet multicast address.

Because of the feature, that a multicast IP packet will be received by any member of a multicast group, sending and receiving of IP multicast packets is only usable with UDP sockets (datagram sockets).

After installing a IP multicast address with this API, the application programmer is able use this address as a destination address when sending datagrams. A UDP socket is able to receive datagrams at the specified multicast address.

The maximum number of supported IP multicast addresses in the IPC@CHIP® is limited to 15. Before installing an IP multicast address, you can find out the corresponding Ethernet multicast address with **Map IP multicast address to Ethernet address** API.

This implementation does not support multicast routing. Sending and receiving multicast datagrams works only at local network.
Using multicast addresses is only possible on the Ethernet interface.

## SC1x3/SC23 Comments

This function supports only IPv4 multicast addressing.

## 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

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 IPv4 address of type unsigned long

### Return Value

DX = 0, AX = 0: Success
DX = -1, AX = -1: IP address entry not found

### SC1x3/SC23 Comments

This function supports only IPv4 multicast addressing.

### Related Topics

API function **ADD_IGMP_MEMBERSHIP** - Delete an IP multicast address entry
API function **MCASTIP_TO_MACADDR** - Map IP multicast address to Ethernet address

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0x92: MCASTIP_TO_MACADDR, Map IP multicast address to Ethernet

Maps an IP multicast address to the corresponding Ethernet address.

### Parameters

*AH*
0x92 (= MCASTIP_TO_MACADDR)

*ES:DI*
Pointer to multicast IPv4 address of type unsigned long

*DS:SI*
Output Parameter: Pointer to a 6 byte array where the generated Ethernet multicast address will be written.

## 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 manner: To map an IP multicast address to a corresponding Ethernet multicast address place the low-order 23 bits of the IP multicast 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

## SC1x3/SC23 Comments

This function supports only IPv4 multicast addressing.

## Related Topics

API function **ADD_IGMP_MEMBERSHIP** - Install an IP multicast address entry
API function **DROP_IGMP_MEMBERSHIP** - Delete an IP multicast address entry

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0x9F: DEV_GET_IP, Get IP address and net mask of user device

This function is used to get the IP address and net mask assigned to a user device.

## Parameters

*AH*
0x9F (=DEV_GET_IP)

*ES:DI*
**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

*DX:SI*
Output Parameter: Pointer to the buffer, where the 32 bit IP address will be stored by this API in **network byte order**.

*BX:CX*
Output Parameter: Pointer to the buffer, where the 32 bit net mask will be stored by this API in **network byte order**.

## Return Value

DX = 0, AX = 0, success
DX = -1, AX contains **error** code

## Comments

This API is used to get the IP address information from a user device interface. This function can be useful for PPP client user defined devices.

The C-library implementation of this function has the prototype:

```
int Dev_Get_IP( DevUserIfaceHandle DevHandlePtr,
                          long * ipaddr,
                          long * netmask,
                          int * errorcode) ;
```

## Related Topics

**DEV_PPP_GET_PEER_IP** - Get PPP peer IP address

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xA0: DEV_OPEN_IFACE, Install user device driver

Open and install a user TCP device driver/interface at the internal TCP/IP stack of the @CHIP-RTOS. Add a new link layer for TCP/IP communication.

## Parameters

*AH*
0xA0 (=DEV_OPEN_IFACE)

*ES:DI*
Pointer to user filled **DevUserDriver** structure, which contains all needed information (device driver functions, etc.) for the user's driver implementation.

## Return Value

DX = 0, AX = 0 (or =236): Success

The `IfaceHandle`member of **DevUserDriver**structure at [ES:DI] contains the valid **Device Handle**, which will be required as a parameter for most of the other device interface API for this device.

If member variables `iface_type`and `use_dhcp`were set to `DEV_ETHERNET`(=1), AX should contain code 236. In this case, the user must make the additional call to **DEV_WAIT_DHCP_COMPLETE**API to wait for completion of the IP configuration by DHCP.

DX = -1: AX contains **error** code. Installation failed

## Comments

This is the first function to use when adding your own interface to the TCP/IP stack of the @CHIP-RTOS.

The TCP/IP API calls 0x9F - 0xB1 allow the application developer to implement an additional TCP/IP driver interface for a connected hardware device (e.g. connected UART or an Ethernet controller). This new interface has its own IP configuration and is used by the TCP/IP stack for IP communication in the same way as the pre-installed internal devices of the IPC@CHIP® (e.g. Internal Ethernet, PPP server or PPP client).

For more detailed description and a generic example see **How to add an TCP/IP device driver/link layer interface**.

On the first DEV_OPEN_IFACE call for installing a TCP/IP device driver, the `IfaceHandle` member of the **DevUserDriver** structure at [ES:DI] must be NULL. If an interface is closed with DEV_CLOSE_IFACE and the user wants to reopen that device interface **(e.g. for changing IP configuration)** , the `IfaceHandle` value set by the system in the first DEV_OPEN_IFACE call must be preserved.

The `DEV_ADD_ONLY` flag bit can be OR'ed into the `iface_type` member of **DevUserDriver** structure in order to add the new device interface to the TCP/IP stack without immediately configuring this new device. In this way, PPP options may be set between a first call to DEV_OPEN_IFACE which adds a new interface and a subsequent call to configure the interface.

You will find all needed types, constants and prototypes in the CLIB header files `TCPIPAPI.H` and `TCPIP.H`.

The C-library implementation of this function has the prototype:

```
int Dev_Open_Interface( DevUserDriver far * DriverInfo,
                        int * errorcode)
```

## SC1x3/SC23 Comments

At the current SC1x3/SC23 CHIP-RTOS version, the device driver interface only supports IPv4.

The SC1x3 CHIP-RTOS also supports the PPPoE client interface (`DEV_PPPoE_CLIENT`).

## Related Topics

**DEV_CLOSE_IFACE** - Close device driver interface
**DEV_WAIT_DHCP_COMPLETE** - Wait for DHCP IP configuration
**Example** - Installing device driver

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xA1: DEV_CLOSE_IFACE, Close TCP/IP device driver/interface

Close a TCP device driver/interface, remove an added link layer from the internal TCP/IP stack.

## Parameters

*AH*

0xA1 (=DEV_CLOSE_IFACE)

*ES:DI*

**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

## Return Value

DX = 0, AX = 0: Success.
DX = -1: AX contains **error** code
222: Invalid parameter
236: Closing already in progress
237: Device already closed

## Comments

This function closes the device driver/interface. It calls the driver close function and deactivates the layer from the internal TCP/IP stack of the @CHIP-RTOS.

For more detailed description see **How to add a user device driver/link layer interface**

The C-library implementation of this function has the prototype:

```
int Dev_Close_Interface(DevUserIfaceHandle DevHandlePtr,
                        int * errorcode);
```

## SC1x3/SC23 Comments

At the current SC1x3/SC23 CHIP-RTOS version, the device driver interface only supports IPv4.

## Related Topics

**DEV_OPEN_IFACE** - Open device driver interface

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xA2: DEV_RECV_IFACE, Move received data

Receive and process incoming data at the TCP/IP stack.

## Parameters

*AH*

0xA2 (=DEV_RECV_IFACE)

*ES:DI*

**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

## Return Value

DX = 0, AX = 0: Success.
DX = -1: Failure, AX contains **error** code
255: Insufficient memory to complete operation

## Comments

This function must be called to process incoming data on the TCP/IP stack. This function in turn calls the driver receive function (specified by the `DevRecv` member of the **DevUserDriver** structure at the **DEV_OPEN_IFACE** call) and inserts the received data into the receive queue of the matching socket (if there is one). Any tasks waiting on the socket (sleeping at **RECV** or **RECVFROM**) are then signaled that there is new data available. Only IGMP and ARP requests are directly processed in the task which does this call.

**Do not call this API from inside of an ISR!** This function should be used only in combination with the **Wait for receive event** API call in the user's receiver task for their device driver interface.

For more detailed description see **How to add an TCP/IP device driver/link layer interface**

The C-library implementation of this function has the prototype:

```
int Dev_Recv_Interface( DevUserIfaceHandle DevHandlePtr,
                        int * errorcode);
```

## Related Topics

**Example** - Receiver Task which calls this API
Example **Receiver** function which would be called by this API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xA3: DEV_RECV_WAIT, Wait for received data

Waits for data received signal from Interrupt Service Routine.

## Parameters

*AH*

0xA3 (=DEV_RECV_WAIT)

*ES:DI*

**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the

DEV_OPEN_IFACE call.

## Return Value

DX = 0, AX = 0: Success
DX = -1: Failure, AX contains error code
255: Insufficient memory to complete operation

## Comments

This function sleeps until a DEV_NOTIFY_ISR signal by the device's ISR indicates that received data is available. This function should be used in combination with the DEV_RECV_IFACE API within the user implemented receiver task.

For more detailed description see How to add a user device driver/link layer interface

The C-library implementation of this function has the prototype:

```
int Dev_Recv_Wait(DevUserIfaceHandle DevHandlePtr,
                  int * errorcode);
```

## Related Topics

Example - Receiver Task

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xA4: DEV_NOTIFY_ISR, Signal from ISR

Notify the TCP/IP stack from inside of an Interrupt Service Routine that there is incoming data available and/or the device has sent a frame successfully.

## Parameters

*AH*
0xA4 (=DEV_NOTIFY_ISR)

*BX*
Number of available received packets

*CX*
Number of sent packets completed (provisional)

*ES:DI*
Device handle from the `IfaceHandle` member of the `DevUserDriver` structure used at the DEV_OPEN_IFACE call.

## Return Value

DX = 0, AX = 0

## Comments

This function should only be called from the device driver specific ISR. It can be used to signal a receiver task which is waiting at the **DEV_RECV_WAIT** call. In this case the number of received packets should be set in BX register.

The current implementation does not support a transmit task. The sent packets count in CX is a provision for informing a transmit task about completion of packet transmission.

For more detailed description see **How to add a user device driver/link layer interface**.

The C-library implementation of this function has the prototype:

```
int Dev_Notify_ISR( DevUserIfaceHandle DevHandlePtr,
                    unsigned int rcvdPackets,
                    unsigned int sentPackets);
```

## Related Topics

**Example** - User device ISR

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xA5: DEV_GET_BUF, Get a buffer from TCP/IP stack

Get a buffer from TCP/IP stack's pre-allocated buffer for storing incoming data from this device.

## Parameters

*AH*
0xA5 (=DEV_GET_BUF)

*CX*
Size of buffer required, number of bytes

*ES:DI*
Output Parameter: Pointer to a variable of type **DevUserBufferHandle** where the user buffer handle is stored. (This value is for internal use.)

## Return Value

DX: 0, AX 0: Success. Returns in DS:SI a pointer to the beginning of the allocated buffer
DX: -1, AX-1, DS=SI=0 (null pointer)

## Comments

This call should only be used inside the driver specific receive function (implemented by the user). It allocates a buffer from the internal TCP/IP memory pool which can be used for storing the incoming data.

The usage of this function is optional. It is not required, but is recommended that a user work with these buffers from the TCP/IP memory pool.

For more detailed description see **How to add an TCP/IP device driver/link layer interface**

The C-library implementation of this function has the prototype:

```
int Dev_Get_Buffer( DevUserBufferHandle BufHandlePtr,
                    unsigned char far * far * buf,
                    unsigned int len);
```

## Related Topics

Example **Receiver** function
`DevUserBufferHandle` **type** definition
IPC@CHIP® **TCP/IP memory** `chip.ini` Configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xA6: DEV_SND_COMPLETE, Signal message send complete

This API signals the TCP/IP stack that the final frame in a message has been sent. (Note that a message can be fragmented into multiple link layer frames.)

## Parameters

*AH*
0xA6 (=DEV_SND_COMPLETE)

*ES:DI*
**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

## Return Value

DX = 0, AX = 0

## Comments

This function must be called from the driver send function (implemented by the user) on completion of sending data. After this call, the TCP/IP stack of the @CHIP-RTOS is able to reuse the transmission data buffer.

For more detailed description see **How to add an TCP/IP device driver/link layer interface**

The C-library implementation of this function has the prototype:

```
int Dev_Send_Complete(DevUserIfaceHandle DevHandlePtr);
```

## Related Topics

**Example** - User's Device Send Function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xA7: DEV_WAIT_DHCP_COMPLETE, Wait for DHCP

Waits for completion of IP configuration process by DHCP *("Dynamic Host Configuration Protocol")* . This call must be used if **DEV_OPEN_IFACE** API was called with DHCP enabled (see **use_dhcp**) in `DevUserDriver` structure.

## Parameters

*AH*

0xA7 (=DEV_WAIT_DHCP_COMPLETE)

*CX*

Timeout seconds. This value will depend on the DHCP server used. A minimum value of 15 seconds is recommended.

*ES:DI*

Pointer to same **DevUserDriver** structure used at the **DEV_OPEN_IFACE** call.

## Return Value

DX = 0, AX = 0: Success

The variables `IPAddr`and `Netmask`in the **DevUserDriver** structure contain the valid IP configuration. The `Dhcp_Data`structure member points to the full **configuration data**provided by the DHCP server.

DX = -1: AX contains **error** code: 260 Timeout.

## Comments

Before calling this function, the device receiver task must be running.

For more detailed description see **How to add an TCP/IP device driver/link layer interface**.

You will find all needed types, constants and prototypes in the CLIB header files `TCPIPAPI.H` and `TCPIP.H`. The C-library implementation of this function has the prototype:

```
int Dev_Wait_DHCP_Complete( DevUserDriver far * DriverInfo,
                            unsigned int time_s,
                            int * errorcode)
```

## Related Topics

**Example** - User's receiver task

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xA8: DEV_PPP_REGISTER_CHAP, Install a CHAP authentication callback function.

When acting as an PPP server, a function must be called to validate an incoming authentication request. The prototype of the user provided function is defined to be:

```
char huge * myCHAPAuthenticate(char *username) ;
```

which returns the password (secret) corresponding to this user. If the user name is invalid, this callback function must return a NULL pointer.

## Parameters

*AH*

0xA8 (=DEV_PPP_REGISTER_CHAP)

*ES:DI*

**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

*DS:SI*

Pointer to the authentication callback function.

## Return Value

DX = 0, AX = 0, success
DX = -1, AX = 222, Invalid driver handle specified

## Comments

The C-library implementation of this function has the prototype:

```
int Dev_PPP_Register_Chap( DevUserIfaceHandle DevHandlePtr,
                           const void * AuthCallbackFunction,
                           int * errorcode) ;
```

## SC1x Comments

This function is only available in @CHIP-RTOS variants with the PPP feature included.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xA9: DEV_PPP_REGISTER_PAP, Install a PAP authentication callback function

When acting as an PPP server, a function must be called to validate an incoming authentication request. The prototype of the user provided function is defined to be:

```
int huge * myPAPAuthenticate(char *username,char * password) ;
```

which returns 1 for a valid username/password pair, else 0.

## Parameters

*AH*

0xA9 (=DEV_PPP_REGISTER_PAP)

*ES:DI*

Device handle from the `IfaceHandle` member of the `DevUserDriver` structure used at the DEV_OPEN_IFACE call.

*DS:SI*

Pointer to the authentication callback function.

## Return Value

DX = 0, AX = 0, success
DX = -1, AX = 222, Invalid driver handle specified

## Comments

The C-library implementation of this function has the prototype:

```
int Dev_PPP_Register_Pap( DevUserIfaceHandle DevHandlePtr,
                          const void * AuthCallbackFunction,
                          int * errorcode) ;
```

## SC1x Comments

This function is only available in @CHIP-RTOS variants with the PPP feature included.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0xAA: DEV_PPP_SET_OPTION, Set PPP options

This function is used to set the PPP options, that we want to negotiate as well those options that we will allow.

### Parameters

*AH*
0xAA (=DEV_PPP_SET_OPTION)

*ES:DI*
**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

*DS:SI*
Pointer to user filled **PPP_Option** structure, which contains all needed option parameters.

### Return Value

DX = 0, AX = 0, success
DX = -1, AX contains **error** code

### Comments

The C-library implementation of this function has the prototype:

```
int Dev_PPP_Set_Option( DevUserIfaceHandle DevHandlePtr,
                        PPP_Option * ppp_option,
                        int * errorcode) ;
```

### SC1x Comments

This function is only available in @CHIP-RTOS variants with the PPP feature included.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0xAB: DEV_PPP_GET_PEER_IP, Get PPP peer IPv4 address

This function is used to get the PPP address, that the remote peer has used.

### Parameters

*AH*

0xAB (=DEV_PPP_GET_PEER_IP)

*ES:DI*

**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

*DS:SI*

Output Parameter: Pointer to the buffer, where the peer address will be stored by this API.

## Return Value

DX = 0, AX = 0, success
DX = -1, AX contains **error** code

## Comments

If a default gateway should be added for that interface, then the retrieved IP address should be used.

The C-library implementation of this function has the prototype:

```
int Dev_PPP_Get_Peer_IP( DevUserIfaceHandle DevHandlePtr,
                         long * Address,
                         int * errorcode) ;
```

## SC1x Comments

This function is only available in @CHIP-RTOS variants with the PPP feature included.

## Related Topics

**DEV_GET_IP** - Get user device interface IP address

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xAC: DEV_PPP_GET_PEER_DNS, Get PPP peer address

This function is used to return the DNS addresses as negotiated by the remote PPP server.

## Parameters

*AH*

0xAC (=DEV_PPP_GET_PEER_DNS)

*ES:DI*

**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

*BX*

Flag 1: Primary DNS address 2: Secondary DNS address

*DS:SI*

Pointer to the buffer, where the DNS address will be stored into.

## Return Value

DX = 0, AX = 0, success
DX = -1, AX contains **error** code

## Comments

The C-library implementation of this function has the prototype:

```
int Dev_PPP_Get_Peer_DNSIP( DevUserIfaceHandle DevHandlePtr,
                            long * Address,
                            int flag,
                            int * errorcode) ;
```

## SC1x Comments

This function is only available in @CHIP-RTOS variants with the PPP feature included.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xAD: DEV_GET_HANDLE_BYNAME, Get driver handle pointer

This function returns the driver handle pointer by a given driver name

## Parameters

*AH*

0xAD (=DEV_GET_HANDLE_BYNAME)

*DS:SI*

Driver name

*ES:DI*

Output Parameter: Driver handle pointer **Device handle**

## Return Value

DX = 0, [ES:DI] is driver handle pointer
DX = -1, Driver not found, es=di=0

## Comments

This function delivers the required driver handle pointer, which is necessary for using **DEV_CONFIG_IFACE**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xAE: DEV_CONFIG_IFACE, Configure an interface

The function provides the IPv4 configuration of an interface. This functions provides the possibility to configure a device interface (e.g. the standard internal ethernet interface) with multiple IP addresses by enter a particular multihome index number.

## Parameters

*AH*

0xAE (=DEV_CONFIG_IFACE)

*ES:DI*

Pointer to **DevIpv4IfaceCfg** with IP configuration parameters provided by the caller

## Return Value

DX = 0, AX = 0
DX = -1, failure AX contains errorcode, ES=DI=0

## Comments

If an interface is already configured at the specified multihomeindex number, the user first has to unconfigure it with **DEV_UNCONFIG_IFACE**. DHCP configuration of a device is only possible at multihomeindex 0.
It is recommended to use this function only for changing ip configuration of own device interfaces or for changing IP configuration of the internal ethernet interface for multihomeindex 1..3 (SC1x3/SC23 1 ..4). Changing the IP configuration of the ethernet device at index 0 should be done by the commands **IP** and **IPETH** because of the automatic actualization of the settings at chip.ini entry **ADDRESS**.
Clibrary-based example:

```
DevUserIfaceHandle DevHandlePtr;
DevIpv4IfaceCfg  devcfg;
int error;
//Get the driver handle by name
if(-1==Dev_Get_Handle_By_Name("NE2000",&DevHandlePtr,&error))
{
  printf("\r\nError: Cannot detect driver handle, code %d",error);
  return 0;
}
//Set ip configuration
devcfg.IfaceHandle    = DevHandlePtr;
devcfg.flags          = 0;
devcfg.multiHomeIndex = 1;  //configure particular multihomeindex 1
inet_addr("192.168.200.59",&devcfg.IpAddr);
inet_addr("255.255.255.0"",&devcfg.Netmask)

if(-1==Dev_Config_Iface(&devcfg,&error))
{
     printf("Error %d\r\n",error);
}
```

## Related Topics

**DEV_GET_HANDLE_BYNAME** - Get driver handle by name
**DEV_UNCONFIG_IFACE** - Unconfigure an interface

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xAF: DEV_CONFIG_IFACE_IPV6, IPv6 configuration of an interface

The function provides the IPv6 configuration of an interface. This functions provides the possibility to configure a device interface (e.g. the standard internal ethernet interface) with multiple IP addresses by enter a particular multihome index number.

## Parameters

*AH*
0xAF (=DEV_CONFIG_IFACE_IPV6)

*ES:DI*
Pointer to **DevIpv6IfaceCfg** with IPv6 configuration parameters provided by the caller

## Return Value

DX = 0, AX = 0
DX = -1, failure AX contains errorcode, ES=DI=0

## Related Topics

**DEV_CONFIG_IFACE** - IPv4 Configuration of an interface

DEV_UNCONFIG_IFACE - Unconfigure an interface

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xB0: DEV_UNCONFIG_IFACE, Unconfigure an interface

The function is used to remove an IPv4/v6 configuration from an interface for a particular multi home index

## Parameters

*AH*

0xB0 (=DEV_UNCONFIG_IFACE)

*AL*

Multihome index number

*BX*

Protocol family (AF_INET or AF_INET6)

*ES:DI*

Device handle

## Return Value

DX = 0, AX = 0
DX = -1, failure AX contains errorcode=0

## Comments

It is recommended to use this function only to unconfigure own device interfaces or to unconfigure the IP address of the internal ethernet interface for multihomeindex 1-3 (SC1x3/SC23 1-4).
Changing the IP configuration of ethernet interface at index 0 should be done by the commands IP and IPETH.

## Related Topics

DEV_CONFIG_IFACE - IPv4 Configuration of an interface
DEV_CONFIG_IFACE_IPV6 - IPv6 Configuration of an interface

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0xB1: DEV_PPP_GET_PEER_IPV6, Get PPP peer IPv6 address

This function is used to get the PPP address, that the remote peer has used.

### Parameters

*AH*

0xB1 (=DEV_PPP_GET_PEER_IPV6)

*ES:DI*

**Device handle** from the `IfaceHandle` member of the `DevUserDriver` **structure** used at the **DEV_OPEN_IFACE** call.

*DS:SI*

Output Parameter: Pointer to the buffer, where the peer address will be stored by this API from type **struct sockaddr_storage**.

### Return Value

DX = 0, AX = 0, success
DX = -1, AX contains **error** code

### Comments

If a default gateway should be added for that interface, then the retrieved IP address should be used.

### SC1x Comments

This function is only available in @CHIP-RTOS variants with the PPP feature included.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0xC0: GET_IPV6_IFACE_ENTRIES, Read IPv6 information for each interface

This function return the IPv6 configuration for each device interface of the TCP/IP stack.

### Parameters

*AH*

0xC0 (=GET_IPV6_IFACE_ENTRIES)

*ES:DI*

Output Parameter: Pointer to a user provided array of `IfaceIPv6_Entry` data **structures** where this API will store the information.

*CX*

Number of data structures in array at [ES:DI]. Up to this number of device interfaces will be reported.

## Return Value

DX = 0, the user provided array at [ES:DI] contains the current device interface information
AX = Number of entries reported in array at [ES:DI].

## Related Topics

**IPv4/v6** - Dual layer stack

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xC1: INET_PTON, Convert an IP address text to binary

Convert an ASCII IPv4 or IPv6 address string from standard text representation to binary.

## Parameters

*AH*

0xC1 (=INET_PTON)

*BX*

Address family, AF_INET (for IPv4) or AF_INET6 (for IPv6)

*ES:DI*

Input Parameter: Pointer to the ASCII IP address text (null terminated) to be converted

*DS:SI*

Output Parameter: Storage for the converted binary address, provided by the user. Buffer must provide 4 bytes for AF_INET and 16 bytes for AF_INET6.

## Return Value

DX = 1, AX=0: Success, the user provided array at [DS:SI] contains the converted address.
DX = 0: Invalid input IPv4 or IPv6 address text at [ES:DI].
DX = -1: BX was set to an invalid address family.

## Comments

The binary IP address is output in network byte order.

## Related Topics

IPV6 - Socket interface
Convert IPv4 address string to binary
Convert binary IPv4 or IPv6 address to ASCII string

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xC2: INET_NTOP, Convert an IP binary address to text

Convert an IPv4 or IPv6 address from the binary format to standard text representation.

## Parameters

*AH*
0xC2 (=INET_NTOP)

*BX*
Address family, AF_INET (for IPv4) or AF_INET6( for IPv6)

*ES:DI*
Points to the IP address in binary format, in network byte order.

*DS:SI*
Storage for the converted address in text format, provided by the user.

*CX*
Size in bytes, of the provided buffer at [DS:SI].
Buffer should provide 16 bytes for AF_INET and 46 Bytes for AF_INET6.

## Return Value

DX = 0, AX=0: Success, the user provided array at [DS:SI] contains the converted address.
DX =-1 AX=-1: Failure, Invalid address family value or inadequate buffer size specified.

## Related Topics

IPV6 - Socket interface
Convert binary IPv4 address to ASCII string
Convert ASCII IPv4 or IPv6 address string to binary

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0xC3: API_OPENSOCKET_IPV6, open an IPv6 socket

Creates an IPv6 end-point for communication and returns a socket descriptor (i.e. a handle).

### Parameters

*AH*

0xC3 (= API_OPENSOCKET_IPV6)

*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

### Comments

This function provides the BSD socket() functionality.

### Related Topics

Open IPv4 socket **API_OPENSOCKET**
Close socket **API_CLOSESOCKET**
**IPV6** - Socket interface

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0xCF: IPSEC_VECTORS, IP Security API

Accesses the set of API that apply to IP Security.

### Parameters

*AH*

0xCF (= IPSEC_VECTORS)

*AL*

Sub selector for one of the IP Security API **services**.

## Comments

This AH vector leads to the family of IP Security API.

## Related Topics

**IP Security** API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xD0: PKI_CERT_ADD, Add a certificate

Add a certificate to the public key infrastructure. The CA certificate must be added before the certificate is added. In addition, the private/public key pair for a local certificate must be added before adding the local certificate.

## Parameters

*AH*
0xD0 (= PKI_CERT_ADD)

*ES:DI*
pointer to **PKI_CERT_ADD_Entry** structure

## Return Value

DX = 0 success
DX != 0 AX: contains **error** code

## Related Topics

Delete certificate **PKI_CERT_DEL**
Add private/public key pair **PKI_OWNKEYPAIR_ADD**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

## Interrupt 0xAC service 0xD1: PKI_CERT_DEL, Delete a certificate

Delete a certificate from the public key infrastructure.

## Parameters

*AH*
0xD1 (= PKI_CERT_DEL)

*ES:DI*
pointer to certificate identification string

*BX*
certificate type
Bit0 = 1: local certificate
Bit1 = 1: non local certificate (CA or peer)
Bit3 = 1: root CA certificate

## Return Value

DX = 0 success
DX != 0 AX: contains **error** code

## Related Topics

Add certificate **PKI_CERT_ADD**
Add private/public key pair **PKI_OWNKEYPAIR_ADD**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xD2: PKI_OWNKEYPAIR_ADD, Add a public/private key pair

Add a public/private key pair for a local certificate to the public key infrastructure.

## Parameters

*AH*
0xD2 (= PKI_OWNKEYPAIR_ADD)

*ES:DI*
pointer to **PKI_OWNKEYPAIR_ADD_Entry** structure

## Return Value

DX = 0 success
DX != 0 AX: contains **error** code

## Related Topics

Add certificate **PKI_CERT_ADD**
Delete certificate **PKI_CERT_DEL**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xD5: SSL_GET_CONNSTATE, Return internal SSL status information

Returns internal SSL status information, required by the WLAN WL01 driver for execution of EAP-TLS protocol.

## Parameters

*AH*
0xD5 (= SSL_GET_CONNSTATE)

*BX*
SSL socket descriptor

## Return Value

DX = 0 success, CX:DI points to the required internal SSL status information
DX != 0 AX: contains **error** code

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.17 | V1.17 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xD6: SSL_HANDSHAKE_COMPLETE, Test SSL handshake status

Determine if an SSL data socket is ready to send and receive data.

## Parameters

*AH*
0xD6 (= SSL_HANDSHAKE_COMPLETE)

*BX*
Socket descriptor

*CX*

Time out in milliseconds:
Set to 0xFFFF (= -1) to block forever.
Set to 0 for non-blocking call.
Otherwise maximum number of milliseconds to wait, up to 65534 ms maximum.

## Return Value

DX = 0
AX = SSL handshake status:
-3: Handshake failure
-2: Invalid socket descriptor
-1: Not a TCP type socket
0: Either not an SSL type socket or handshake in progress
1: SSL handshake completed

## Comments

Use of this function is optional. It allows the user to determine whether or not the SSL handshake process has completed and that the data socket is now ready for data transmission and reception.

This API can be used in either polling mode (CX = 0) or blocking mode. The timeout period is specified in CX. Blocking mode will return as soon as the status indication returned in AX is non-zero, or the specified time-out period has elapsed. The CX register may be set to 0xFFFF to block for an indefinite amount of time, in which case AX register is guaranteed to be non-zero on return.

Alternatively, the user can simply make repeated calls to the **send()** and/or **recv()** functions until this handshake is completed. The SSL handshake is performed in the user's thread, so one of these three functions must be called in order to complete the SSL connection.

## Beck C-Library Usage

**SSL_HandshakeComplete()**

## Related Topics

send() **API_SEND**
recv() **API_RECV**
New SSL session **SSL_SESSION_NEW**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.10 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xD7: SSL_SESSION_CLOSE, Close a SSL session

Call this function to close a SSL session. All cached connection information in this session will be freed too. User should close the SSL connection or the socket, first.

## Parameters

*AH*
0xD7 (= SSL_SESSION_CLOSE)

*BX*
Session ID

## Return Value

DX = 0 success
DX != 0 AX: contains **error** code

## Comments

Note that the peer may not accept any data after the SSL layer is closed.

## Related Topics

Close socket **API_CLOSESOCKET**
Close SSL connection **SSL_CONN_CLOSE**
New SSL session **SSL_SESSION_NEW**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xD8: SSL_SESSION_NEW, Open a new SSL session

Open a new SSL session. Could be used as either SSL server session or SSL client session. If it is used as a server session the field certIdentity should be non-null, otherwise the SSL server can't run on this session.

## Parameters

*AH*
0xD8 (= SSL_SESSION_NEW)

*ES:DI*
pointer to **SSL_SESSION_NEW_Entry** structure

## Return Value

DX = 0 success AX: Session ID
DX != 0 AX: contains **error** code

## Comments

Each session maintains its own connect information for the convenience of resumption. You can resume a previous connection only if the current connect attempt and the previous one are in the same SSL session, and your previous connection cache hasn't been overwritten yet.

For SSL server, all connections spawned from a single listening socket belong to one session, you should assign the session ID to the listening socket. All accepted sockets will inherit that session ID.

For SSL client, user determines which session one connection belongs to. For example, users may decide that any connection to the same SSL server uses a single session ID, so that when user is trying to connect to the same server again, he may resume the previous connection.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xD9: SSL_SESSION_PROP, Set proposals for a SSL session

Use this function to set proposals for a SSL session. For a client session, the proposals are used to construct the Client Hello message. For a server session, the proposals are used to match Client Hello message in order to determine the final cipher suite to use.

### Parameters

*AH*
0xD9 (= SSL_SESSION_PROP)

*BX*
Session ID

*ES:DI*
pointer to proposals array (array of int)

*CX*
number of entries in the proposal array

### Return Value

DX = 0 success
DX != 0 AX: contains **error** code

### Comments

To use this function on a server session, a previous call to function **SSL_SESSION_OPT** is needed. If the user doesn't call this function to set any session proposals, the default value is going to be used.

**Possible proposals array entries:**

```
TLS_RSA_NULL_MD5           = 0x0001
TLS_RSA_NULL_SHA           = 0x0002
TLS_RSA_EXPORT_RC4_40_MD5 = 0x0003
TLS_RSA_RC4_128_MD5        = 0x0004
TLS_RSA_RC4_128_SHA        = 0x0005
TLS_RSA_3DES_EDE_CBC_SHA   = 0x000A
TLS_RSA_AES_128_CBC_SHA    = 0x002F
TLS_RSA_AES_256_CBC_SHA    = 0x0035
TLS_RSA_EPT1K_RC4_56_SHA   = 0x0064
```

## Related Topics

Set SSL session options **SSL_SESSION_OPT**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xDA: SSL_SESSION_OPT, Set options for a SSL session

Call this function to set SSL session options.

## Parameters

*AH*

0xDA (= SSL_SESSION_OPT)

*BX*

Session ID

*DX*

option
- 1: Client Authentication
- 2: Server Proposals

*CX*

option value
- 0: disable option
- 1: enable option

## Return Value

DX = 0 success
DX != 0 AX: contains **error** code

## Comments

The client authentication option is used to enable client authentication on a SSL server session. It takes effect only if the session is bound to an SSL server socket, in which case, upon receiving the ClientHello message, the SSL server is going to send back a CertificateRequest message along with its own

Certificate, ServerHello and ServerHelloDone messages.

The server proposals option is used to allow the SSL server to have more control on which cipher suite to use. If this option is not used on an SSL server session, upon receiving a ClientHello message, the SSL server just chooses the first matched cipher suite from its own session proposal list, according to server's preference. If a cipher suite proposed by the client is not the SSL server session's proposal list, even if the SSL server supports that cipher suite, it won't be chosen. For example, if server proposals option is enabled and server prefers cipher suites 1,2,3,4 and in this order. However, client only offers 4,3,2 and in this order, then cipher suite 2 will be chosen.

## Related Topics

Set SSL session proposals **SSL_SESSION_PROP**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xDB: SSL_CLI_CA_ADD, Add a client CA certificate

Call this function to add client CAs to ServerHello message

## Parameters

*AH*
0xDB (= SSL_CLI_CA_ADD)

*BX*
Session ID

*ES:DI*
Pointer to client CA certificate identity name

## Return Value

DX = 0 success
DX != 0 AX: contains **error** code

## Comments

If client authentication option is used to enable client authentication on a SSL server session, this call can be used to add CAs to the ServerHello message (called CertificateRequest).
This function must be called against an SSL server session. The SSL server will list CA's Distinguished Name to the client in his ServerHello message . This function can be called multiple times to add multiple potential CAs for client certificates.
Before you can use this call, the client CAs certificate must be added to the PKI.

## Related Topics

Add certificate **PKI_CERT_ADD**

Set SSL session options **SSL_SESSION_OPT**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xDC: SSL_CLI_START, Start the SSL client

Call this function to start the SSL client and SSL handshaking. Before this call, the user should set proper socket options on socket descriptor.

## Parameters

*AH*

0xDC (= SSL_CLI_START)

*BX*

Socket descriptor

*ES:DI*

Pointer to server common name.
The common name is used to verify the certificate of the peer. If user passes in a NULL string, the certificate identity is not checked. This means, any valid certificate, even if it belongs to a bad party, will be accepted.

## Return Value

DX = 0 success
DX != 0 AX: contains **error** code

## Comments

The certificate identity (common name field) may contain the wildcard character which is considered to match any single domain name component or component fragment. E.g., "*.a.com" in certificate Common Name matches the server common name "foo.a.com" but not "bar.foo.a.com". And certificate common name "f*.com" matches server common name "foo.com" but not "bar.com".

## Related Topics

Set socket options **API_SETSOCKOPT**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**

Index page

## Interrupt 0xAC service 0xDD: SSL_SEND_FLUSH, Queued user data will be flushed

SSL will queue user data if the send path is not triggered up when socket option TCP_SSL_SEND_MIN_SIZE is greater than zero. All data will be sent out rigth away, if this function is called.

### Parameters

*AH*
0xDD (= SSL_SEND_FLUSH)

*BX*
Socket descriptor

### Return Value

DX = 0 success, AX contains number of bytes flushed
DX != 0 AX: contains **error** code

### Related Topics

Set socket options **API_SETSOCKOPT**

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

## Interrupt 0xAC service 0xDE: SSL_CONN_CLOSE, Close a SSL connection

The user data will be flushed and a close notify message will be send to the SSL peer. User may also close the whole socket instead of only closing the SSL layer.

### Parameters

*AH*
0xDE (= SSL_CONN_CLOSE)

*BX*
Socket descriptor

### Return Value

DX = 0 success
DX != 0 AX: contains **error** code

### Comments

Note that the peer may not accept any data after the SSL layer is closed.

## Related Topics

Close socket **API_CLOSESOCKET**
Close SSL session **SSL_SESSION_CLOSE**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xDF: CRYPT_HASH, Build MD5/SHA1 checksum

The user can use this function to create MD5/SHA1 checksums

## Parameters

*AH*
0xDF (= CRYPT_HASH)

*AL*
Mode
0x01: Init MD5 hash function
0x02: Update the MD5 hash
0x03: Finalize the MD5 hash

**SHA1 hash is only available on SC1x3/SC23:**
0x04: Init SHA1 hash function
0x05: Update the SHA1 hash
0x06: Finalize the SHA1 hash

*ES:DI*
Pointer to **MD5 context structure** or **SHA1 context structure**

*DX:SI*
Pointer to input buffer (valid for mode="update" only)

*CX*
length of input buffer (valid for mode="update" only)

## Return Value

DX = 0 success
DX != 0 error

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.11 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xE0: CRYPT_CIPHER, Use symmetric crypto functions

The user can use this function to encrypt/decrypt messages

## Parameters

*AH*

0xE0 (= CRYPT_CIPHER)

*AL*

Mode

0x01: Make AES encrypt key
0x02: Make AES decrypt key
0x03: Encrypt AES message
0x04: Decrypt AES message

*ES:DI*

In mode 0x01 and 0x02: Pointer to **GenericKey structure**
In mode 0x03 and 0x04: Pointer to input buffer

*DX:SI*

Pointer to **AES context structure**

*CX:BX*

In mode 0x03 and 0x04: Pointer to output buffer

## Return Value

DX = 0 success
DX != 0 error

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

Top of list
Index page

---

End of document

---

## AddDefaultGatewayEx

Add the default gateway (Extended).

```
int AddDefaultGatewayEx ( IP_INTERFACE which_interface,
                          unsigned long far *Gateway,
                          void far *userIfaceHandle,
                          int *error );
```

## Parameters

*which_interface*

Enumeration type:

- II_ETHERNET = 0: Ethernet
- II_PPP_SERVER = 1: PPP server
- II_PPP_CLIENT = 2: PPP client
- II_USER_DEVICE = 3: user device driver

*Gateway*

Pointer to gateway IP address in network byte order.

*userIfaceHandle*

If `which_interface==3`, Pointer to user device handle.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## Comments

If this function is used, the **gateway** entry in the `CHIP.INI` becomes invalidated, but unchanged. If the PPP server or PPP client is specified by the `which_interface` parameter, then the gateway is set to the remote peer IP address.

## See Also

AddDefaultGateway DelDefaultGateway GetDefaultGateway
AddStaticRoute DelStaticRoute

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Related Topics

IP Gateway `CHIP.INI` Configuration
PPP server Gateway `CHIP.INI` Configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---

# @CHIP-RTOS - IPv4/v6 Dual layer stack

## Introduction

### IPv4/v6 Dual Layer Stack

Additional to the IPv4 protocol, the TCP/IP protocol stack of the IPC@CHIP® SC1x3/SC23 provides IPv6.

IPv6 is sometimes called the Next Generation Internet Protocol, or IPng. The current Internet and most of corporate or private intranets use IPv4, developed in 1970s and 1980s. Now with IPv6, the first significant upgrade of the Internet protocol has been started. The compelling reason behind the formation of IPv6 was lack of address space, especially in the heavily populated countries of Asia such as China.

The most important changes from IPv4 to IPv6 can be summarized as follows:

1. **Expanded addressing and autoconfiguration mechanisms**

   The address size for IPv6 is 128 bits (as defined by RFC 2373 and 2374), instead of 32 bits for IPv4. This solves the problem of limited address space and offers much simpler address configuration.

   IPv6 addresses are mostly composed of two logical parts:
   A 64-bit network prefix, and a 64-bit host-addressing part, which is often automatically generated from the interface MAC address.
   IPv6 addresses are normally written as eight groups of four hexadecimal digits. For example, 2001:0db8:85a3:08d3:1319:8a2e:0370:7334 is a valid IPv6 address. If the address is an IPv4 address in disguise, the last 32 bits may be written in decimal thus ::ffff:192.168.89.9.

   Addresses on IPv6 interfaces can be configured without manual intervention. With **stateless address autoconfiguration**, a device interface is automatically assigned a link-local IPv6 address by the system. This link-local IPv6 address is generated by prepending a fixed local address prefix (fe80::) to a token derived from the MAC address. (The address is verified to be unique.) This allows each IPv6 interface to have at least one source address that can be used by Neighbor Discovery. If an IPv6 router on the network advertises network prefixes in router advertisements, IPv6 derives IPv6 addresses based on the network interface identifier of the interface and on the network prefixes advertised.

2. **Simplified new header format**

   The IPv6 header has a length of 40 bytes. Some fields of the IPv4-header are removed. This allows a faster and easier processing of IP packets. The header contains both source and destination addresses (128 bits each), as well as the version (4-bit IP version), traffic class (8 bits, Packet Priority), flow label (20 bits, QoS management), payload length (16 bits), next header (8 bits), and hop limit (8 bits, time to live). The payload can be up to 64k in size in standard mode, or larger with a "jumbo payload" option.

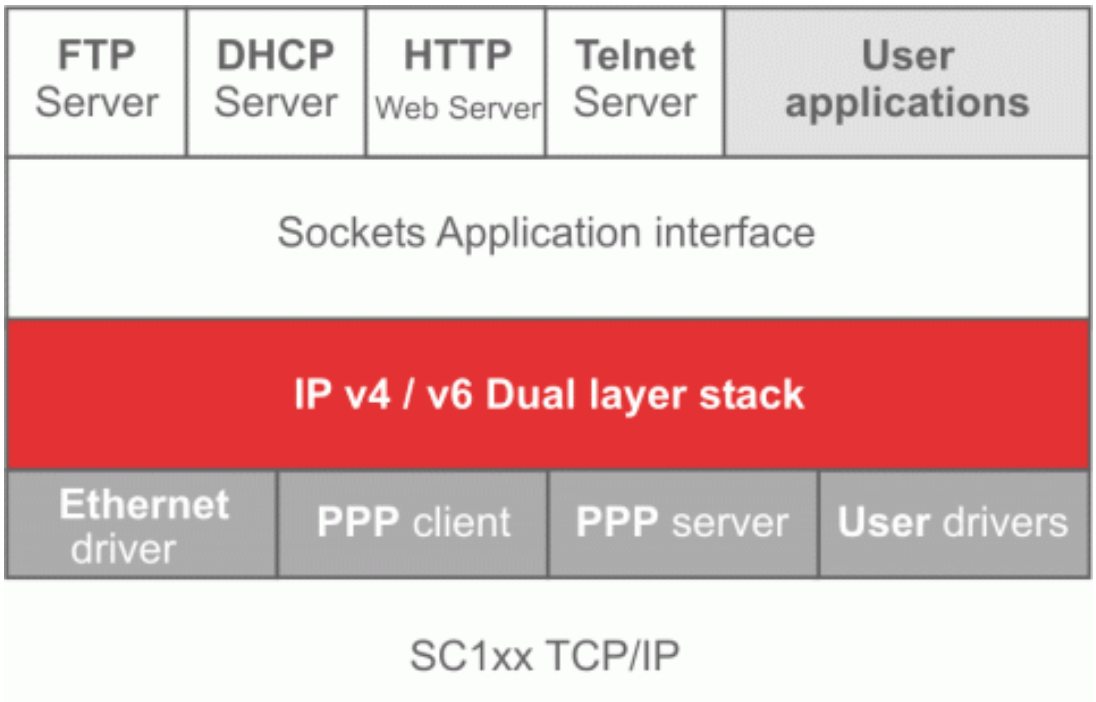
IPv6 header, 40 bytes

| | | |
|---|---|---|
| version | traffic class | flow label |
| payload length | next header | hop limit |
| source address | | |
| destination address | | |

### 3. Better support for extensions and options

In IPv4, all options were integrated into the basis packet header. In IPv6, options are handled with **Extension headers**. Extension headers are optional and only inserted if necessary. For example, the **IPSec**security option is inserted by using an extension header. Extension headers allow a more flexible and efficent construction of IP packets. New options defined in the future can be easily inserted.

The TCP/IP stack of the IPC@CHIP® works as a dual layer stack. If the IPv6 protocol is enabled at the IPC@CHIP®, all TCP/IP applications (e.g HTTP, Telnet or FTP servers) are accessible via both IPv4 and IPv6. The provided socket interface supports both protocol types.




We cannot explain the entire functionality of the IPv6 protocol in this document. The IPv6 base protocol is described in RFC 1883. A good book for understanding IPv6 is "IPv6 essentials" from Silvia Hagen, published by O'Reilly. Under **www.ipv6.org**, you will also find several documents explaining the essentials of IPv6.

**Please note: The IPv6 on IPC@CHIP® is under development and currently not completed. Expansions and modifications of the RTOS and the provided API functions are possible in later versions.**

In the following sections we describe the provided features and functionality of IPv6 offered by the

IPC@CHIP®.

- Enable IPv6 protocol
- Address configuration
- Socket interface

---

## Enable IPv6 protocol

IPv6 on the IPC@CHIP® must be enabled by the chip.ini entry **Ipv6 enable**.

**Top of list**
**Index page**

---

## Address configuration

Configuration of IPv6 addresses is provided as follows:

1. **Stateless autoconfiguration**
2. **Manual configuration**
3. **Get PPP server configuration**
4. **Open PPP client IPv6 link**

The shell command **IPV6CFG** lists the configuration of all started device interfaces.

The TCP/IP API function **GET_IPV6_IFACE_ENTRIES** provides read-only access from within user application programs.

**Top of list**
**Index page**

---

## Socket interface

The socket application interface of the IPC@CHIP® has been updated to add support for IPv6. For IPv6 communication, a socket must be opened with the API function **API_OPENSOCKET_EXT**.

Depending on the API used to open the socket, all subsequent socket API functions, e.g.:

- BIND
- CONNECT
- RECVFROM
- SENDTO
- ACCEPT

use either the **sockaddr_in** or **sockaddr_in6** structure, defined in the CLIB header file tcpipapi.h.

**Important notes:**

1. If a IPC@CHIP® host has a valid IPv4 configuration, it is also possible to communicate over the IPv6-capable socket via the IPv4 address. In such a case the IPv4 addresses are stored internally (at **sockaddr_in6** structure member **sin6_addr** ) as a IPv4-mapped IPv6 address (e.g. 172.30.8.69 is stored as ::FFFF:172.30.8.69).

2. For support of both protocols, the C-library functions **bind, connect, sendto, recvfrom, ...** use a pointer to the generic **sockaddr** structure type. For example the `addressPtr` parameter here:

```
int bind ( int sd, const struct sockaddr far *addressPtr, int *error );
```

3. Conversion of IPv4 and IPv6 addresses from text format to binary and back can be done by using the **INET_PTON** and **INET_NTOP** functions.

4. Provided IPv6 socket API examples (built with Clib functions) are:

- UDPEchoClient, udp6clie.c
- UDPEchoServer, udp6serv.c
- TCPEchoClient, tcp6clie.c
- TCPEchoServer, tcp6serv.c

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

End of document

---

## IPsec_Start

This function activates the IP security function and IKE task within the system's TCP/IP stack.

```
int IPsec_Start(void) ;
```

## Parameters

*-- none --*

## Return Value

error code, zero on success. Otherwise on failure:
less than 5000: Look **here**for error code description.

5000 <= error code <6000: The **POLICY_FILE**restore operation has failed. Subtract 5100 from this value and check **here**for resulting error value.

6000 <= error code <7000: The **PRESHARED_KEY**loading failed. Subtract 6100 from this error code and look **here**for a description of the resulting error code.

## Comments

Alternatively, the IP Security can be started by setting the IPSEC **ENABLE** value in the CHIP.INI or by using the **IKE** START console command.

## See Also

**IPsec_Add_Policy()** **IPsec_Status()** **IPsec_Set_Option()**
**IPsec_Restore_Policy()** **IPsec_IKE_Phase1_Set()**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **IPSEC_VECTORS** for more details.

## Related Topics

IPSEC **ENABLE** from CHIP.INI file
**IKE** command

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## IPsec_Add_Policy

This function appends the list of policies in affect for the IP Security.

```
int IPsec_Add_Policy(const tIPSEC_POLICY_PAIR *pairs,
                     const tIPSEC_POLICY_SELECTOR *selectors,
                     const tIPSEC_POLICY_CONTENT *content,
                     int pair_cnt) ;
```

## Parameters

*pairs*

An array of **tIPSEC_POLICY_PAIR** data structures which map `tIPSEC_POLICY_SELECTOR` structures to their corresponding `tIPSEC_POLICY_CONTENT` structure.

*selectors*

An array of **tIPSEC_POLICY_SELECTOR** that will be used to select IP Security policies to be applied to IP traffic.

*content*

An array of **tIPSEC_POLICY_CONTENT** that specify IP Security policies to be applied to IP traffic.

*pair_cnt*

The number of data structures listed in the array at `pairs`.

## Return Value

**Error code**, zero on success

## Comments

The IP security must be **started** prior to using this API.

The **tIPSEC_POLICY_PAIR** structures at `pairs` are used to connect the members of the other two structure arrays, mapping policy selectors to policy content. Care must be taken to assure that these indexes map to existing structures within these two arrays.

Following return from this API, the user data referenced is no longer required and need not be static. This system has copied this information into its IPsec policy database.

Policies are selected with a search through the database which stops at the first entry whose selector

matches some specified conditions. This search is performed in the reverse order in which policies are added. Newest policies are checked first. Policies are added starting at the top of the list of `pairs`. Consequently, the policy pairs listed last are selected ahead of the policy pairs listed first. So catch all policies for bypass or discard should therefore be listed first.

When this API is called more than once without an intervening **IPsec_Clear_Policy()** call, the policies from the most recent call take precedence over any earlier call's policies.

## See Also

**IPsec_Clear_Policy()** **IPsec_Store_Policy()**
**IPsec_Restore_Policy()**

## RTOS API

This library function invokes a RTOS **software** interrupt.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## IPsec_Store_Policy

Create a binary file containing IP Security policy and/or preshared keys.

```
int IPsec_Store_Policy(
        const char                   *file_name,
        const tIPSEC_POLICY_SELECTOR *selectors,
        unsigned int                  selector_count,
        const tIPSEC_POLICY_CONTENT  *contents,
        unsigned int                  content_count,
        const tIPSEC_POLICY_PAIR     *pairs,
        unsigned int                  pair_count,
        const char           * const *preshared_keys,
        unsigned int                  preshared_key_count);
```

## Parameters

*file_name*

Zero terminated ASCII string specifies the path for the binary data file which will be created by this library function.

*selectors*

Points to an array of `selector_count` **tIPSEC_POLICY_SELECTOR** data structures to be stored in the file that is created.

*selector_count*

Number of data structures in array referenced by `selectors`.

*contents*

Points to an array of `content_count` **tIPSEC_POLICY_CONTENT** data structures to be stored in the file that is created.

*content_count*

Number of data structures in array referenced by `contents`.

*pairs*

Points to an array of `pair_count` **tIPSEC_POLICY_PAIR** data structures to be stored in the file that is created.

*pair_count*

Number of data structures in array referenced by `pairs`.

*preshared_keys*

Pointer to an array of ASCIIZ string pointers. Each ASCII string specifies an IP address and preshared key using the same format as the **PRESHARED_KEY** entries in the CHIP.INI.

*preshared_key_count*

Number of pointers to zero terminated ASCII strings listed in array of pointers at `preshared_keys`.

## Return Value

Error code:

0 - Success, file created without problem.
1 - Unable to create the file.
2 - File write failed.
3 - File exceeded 64K byte limit in size.

## Comments

This function creates a binary file which can be specified in the **CHIP.INI** to restore IP security policies at start up or passed to the **IPsec_Restore_Policy()** at run time.

These policy files must not exceed 64 Kbyte size.

One reason this function has been included in the library is to provide users access to source code which creates the Beck format binary files that hold the IP security policy information, and thereby explicitly documenting this binary file's format. (Note that the source code for these C-library functions is available from Beck.)

## See Also

**IPsec_Add_Policy()** **IPsec_Restore_Policy()**

## RTOS API

This library function uses no @Chip-RTOS API directly. Indirectly the compiler's file I/O C-library functions will access the DOS interrupt **0x21** to produce the file.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

# @CHIP-RTOS - IP Security Data Structures

## Data Structures

Here are the data structures used for the IP Security API. The reader is assumed to have some familiarity with IP Security. For example, the following abbreviations the reader will hopefully already be familiar with:

- AH - "Authentication Header", Reference: RFC-4302
- ESP - "Encapsulating Security Payload", Reference: RFC-4303
- SA - "Security Association", Reference: RFC-4301

IP Security Overview

All constants and data structures are defined in either the IPSEC.H or TCPIPAPI.H header files.

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).

**Contents** :


- tIPSEC_ADDR
- tIPSEC_FILE_HEADER
- tIPSEC_POLICY_CONTENT
- tIPSEC_POLICY_PAIR
- tIPSEC_POLICY_SELECTOR
- tIPSEC_TKE_PHASE1
- tIPSEC_TUNNEL


---

## tIPSEC_ADDR

This data structure is used to specify the IP address information used by the IP Security Policy selectors.

```
typedef struct sIPSEC_ADDR
{
    unsigned char Usage ;

    unsigned char Family ;
    unsigned char IpAddr1[16] ;
    unsigned char IpAddr2[16] ;

    unsigned int  Port ;

} tIPSEC_ADDR ;
```

### Comments

`Usage`
: The lower two bits of this byte provide an enumerator which specifies how this IP address information should be applied. It must be set to one of the following constant values:

  `IPSEC_WILD_ADDR`: This will match all IP addresses. This can be used for catch all remaining packets when applying the `IPSEC_FLAG_BYPASS`policy. The contents of all IP addresses within this structure are ignored when this enumerator is used.

  `IPSEC_SINGLE_ADDR`: This value is used when only a single IP address in `IpAddr1`is provided. The content of `IpAddr2`is ignored in this case.

  `IPSEC_RANGE_ADDR`: This value is used when a range of IP addresses is to be used for policy selection. The address range starts at `IpAddr1`and ends at `IpAddr2`, inclusive.

  `IPSEC_NETMASK_ADDR`: This value is used when a netmask is to be used for policy selection. The `IpAddr1`address specifies the network address and `IpAddr2`specifies the netmask.

  The uppermost two bits (`IPSEC_IP1_DOMAIN`and `IPSEC_IP2_DOMAIN`masks) are used as flags by the BeckIPSec Windows PC policy editor tool to designate that a domain name has been stored in the string pool found at the end of the policy file. These two bits are ignored by the system.

`Family`
: Set to either `AF_INET`for IPv4 or `AF_INET6`for IPv6 addresses.

`IpAddr1`

This is either a four byte IPv4 or 16 byte IPv6 address, in network byte order. For the IPv4 case, the trailing 12 bytes are ignored. Depending on the `Usage`member, this IP address is either void, a single IP address, the start of a range of IP addresses or a network address.

`IpAddr2`
This is either a four byte IPv4 or 16 byte IPv6 address, in network byte order. For the IPv4 case, the trailing 12 bytes are ignored. Depending on the `Usage`member, this IP address is either void, the end of a range of IP addresses or a network mask.

`Port`
Optionally, a specific IP port can be specified for which this security policy should be applied. This port number is *not*in network byte order! Set this member to `IPSEC_WILD_PORT`if selection by port number is not desired.

### Related Topics

API function **IPsec_Set_Policy()** - Install IPsec policies
**tIPSEC_POLICY_SELECTOR** - Data structure

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## tIPSEC_FILE_HEADER

This data structure is used internally by the **IPsec_Store_Policy** API. It is include here in order that a specification of the IP security's binary policy file can be stated. These binary files are used by the **POLICY_FILE** entry in the CHIP.INI file to restore IP security policies at system startup time and by the **IPsec_Restore_Policy()** API at run time.

```
#define IPSEC_SANE_HEADER        (0xA345)
#define IPSEC_FILE_VERSION       (1)

typedef struct sIPSEC_FILE_HEADER
{
    unsigned int  Sanity_Check ;
    unsigned int  File_Version ;
    unsigned int  SelectorCount ;
    unsigned int  ContentCount ;
    unsigned int  PairCount ;
    unsigned int  PresharedKeyCount ;
}
tIPSEC_FILE_HEADER ;
```

### Comments

All data in the file is byte aligned with no padding. Following the header are `SelectorCount` **tIPSEC_POLICY_SELECTOR** data structures.

These are then followed by `ContentCount` **tIPSEC_POLICY_CONTENT** data structures.

These are followed by `PairCount` **tIPSEC_POLICY_PAIR** data structures.

These are followed by `PresharedKeyCount` zero terminated ASCII strings. Each of these strings contains both an IP address and preshared key string concatenated with a semicolon in same format used for the **PRESHARED_KEY** CHIP.INI entries.

If the policy file was created by the BeckIPSec Windows PC tool, then the block of preshared key strings may be followed by a sequence of zero terminated ASCII strings representing IP domain names. This block of information in the file is ignored by the RTOS. The sequence of these domain names match the sequence that the `IPSEC_IP1_DOMAIN` and `IPSEC_IP2_DOMAIN` flags appear in the `Usage` bytes of the **tIPSEC_ADDR** structures and `HAVE_DOMAIN_NAME` flags bits in the `FamilyLocal` and `FamilyRemote` bytes of the **tIPSEC_TUNNEL** structures.

The total file size must not exceed 64 Kbyte.

`Sanity_Check`
This signature must be set to 0xA345 in order for the file to be accepted by the system.

`File_Version`
This entry indicates the format of this file, for compatibility with future changes. The current setting is 1.

`SelectorCount`
This count indicates the number of **tIPSEC_POLICY_SELECTOR**structures. These data structures follow the header immediately.

`ContentCount`
This count indicates the number of **tIPSEC_POLICY_CONTENT**structures. These data structures follow the `tIPSEC_POLICY_SELECTOR`data structures immediately.

`PairCount`
This count indicates the number of **tIPSEC_POLICY_PAIR**structures. These data structures follow the `tIPSEC_POLICY_CONTENT`data structures immediately.

`PresharedKeyCount`
This count indicates the number of IP address/preshared key string combinations. These zero terminated strings follow the `tIPSEC_POLICY_PAIR`data structures immediately. Note that a single zero termination byte is present at the end of each IP address/preshared key specifier. These two fields are delimited with a semicolon, ';'.

### Related Topics

API function **IPsec_Store_Policy()** - Create file
API function **IPsec_Restore_Policy()** - Load IPsec policies
**POLICY_FILE** - CHIP.INI entry
**PRESHARED_KEY** - CHIP.INI entry
**tIPSEC_POLICY_SELECTOR** - Data structure
**tIPSEC_POLICY_CONTENT** - Data structure
**tIPSEC_POLICY_PAIR** - Data structure

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## tIPSEC_POLICY_CONTENT

This data structure is used to specify the content of an IP Security policy. Each `sIPSEC_POLICY_CONTENT` may specify either AH or ESP, but not both. More than one structure is required with both protocols are desired.

```
typedef struct sIPSEC_POLICY_CONTENT    // For user API
{
    // AH | ESP, tunnel|transport, policy|packet determination
    unsigned int            RuleFlags ;
    // ESP may also have authentication algorithm
    unsigned char           AuthAlg ;
    unsigned char           EncryptAlg ;
    unsigned long           LifeSeconds ;
    unsigned long           LifeKbytes ;
    struct sIPSEC_TUNNEL    Tunnel ;
    unsigned int            _spare_ ;  // reserved (set to zero)

}  tIPSEC_POLICY_CONTENT ;
```

### Comments

The tIPSEC_POLICY_CONTENT structure is used to specify the host end points for an IP Security Tunnel.

`RuleFlags`

This is a bit field used to designate various specifics about the policy. One and only one of the following four flags must be set to specify the basic action to be taken on a selected group of IP packets:

`IPSEC_FLAG_AH`: Use Authentication Headers.

`IPSEC_FLAG_ESP`: Apply Encapsulating Security Payload.

`IPSEC_FLAG_BYPASS`: Allow the packets to pass through without any IP security action taken on them.

`IPSEC_FLAG_DISCARD`: Discard any IP packets that match this policy's selectors.

A policy is invalid unless exactly one of the above flags is set.

A further policy option is whether to use IP Security tunnel mode between a local and remote security gateway, or to use simply a transport mode directly between this computer and another host. This decision is specified by one of the following two flags:

`IPSEC_FLAG_TUNNEL`: Selects tunnel mode between two security gateways. When this option is selected, the gateways must be specified using the `Tunnel`member of this data structure.

`IPSEC_FLAG_TRANSPORT`: Selects transport mode.

Finally, there are the following miscellaneous options which can be OR'ed into the `RuleFlags`to avoid sharing SA under various conditions.

`IPSEC_FLAG_RIP_PACKET`: Indicates that each remote IP needs its own SA. Prohibit sharing.

`IPSEC_FLAG_LIP_PACKET`: Indicates that each local IP needs its own SA. Prohibit sharing.

`IPSEC_FLAG_LPT_PACKET`: Indicates that each local port needs its own SA. Prohibit sharing.

`IPSEC_FLAG_PROTO_PACKET`: Indicates that each protocol needs its own SA. Prohibit sharing.

`AuthAlg`
This enumerator selects the "Hash Message Authentication Code" (HMAC). The enumeration values are:

`IPSEC_AALG_NULL`- Null algorithm (ESP only).
`IPSEC_AALG_MD5HMAC`- Use MD5 HMAC algorithm.
`IPSEC_AALG_SHA1HMAC`- Use SHA1 HMAC algorithm.

When the `RuleFlags`specify Authentication Header (AH) then this selection must be either MD5 or SHA1. For ESP, the NULL option is also available to indicate that authentication is not desired for the Encapsulated Security Payload header.

`EncryptAlg`

This enumerator selects the encryption algorithm used with ESP. This field only applies when the IPSEC_FLAG_ESPflag is set. The valid enumeration values are:

IPSEC_EALG_NULL- Null algorithm (transparent, perhaps useful for test).
IPSEC_EALG_DESCBC- Use DES algorithm (weak encryption).
IPSEC_EALG_3DESCBC- Use 3DES algorithm.
IPSEC_EALG_AESCBC- Use AES algorithm with 128 bit key.
IPSEC_EALG_AESCBC_192- Use AES algorithm with 192 bit key.
IPSEC_EALG_AESCBC_256- Use AES algorithm with 256 bit key.

LifeSeconds
Specifies the lifetime in seconds of the IPsec SA's generated according to this policy content. Set this to zero for the default value of 12 hours. Internally, this value will be confined to be within the range 120 seconds minimum up to maximum one year.

LifeKbytes
Specifies the lifetime in kilo-bytes of the IPsec SA's generated according to this policy content. Set this to zero for the default value of 64 Mbyte. Internally, this value will be confined to be within the range 10 kbyte minimum up to 4 Gbyte maximum.

Tunnel
This **structure**is used to specify the security gateway IP addresses for tunnel mode. This field is only used when the IPSEC_FLAG_TUNNELoption flag is set.

### Related Topics

API function **IPsec_Set_Policy()** - Install IPsec policies
**tIPSEC_TUNNEL** - Data structure
**IKE** console command - Lists established SA

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## tIPSEC_POLICY_PAIR

This data structure is used by the **IPsec_Set_Policy()** API to map policy selectors to IP security policies.

```
typedef struct sIPSEC_POLICY_PAIR
{
    unsigned int        iSelectorIndex ;
    unsigned char       bContentIndex ;
    unsigned char       bDirection ;

} tIPSEC_POLICY_PAIR ;
```

### Comments

The selectors and content referred to here are the arguments to the **IPsec_Set_Policy()** API.

iSelectorIndex
This index designates a **tIPSEC_POLICY_SELECTOR**structure within the selectorsarray of structures.

bContentIndex
This index designates a **tIPSEC_POLICY_CONTENT**structure within the contentarray of structures.

bDirection
This value is an enumerator used to designate the direction of IP packet traffic to which this policy should be applied. It can be set to any one of the following three settings.

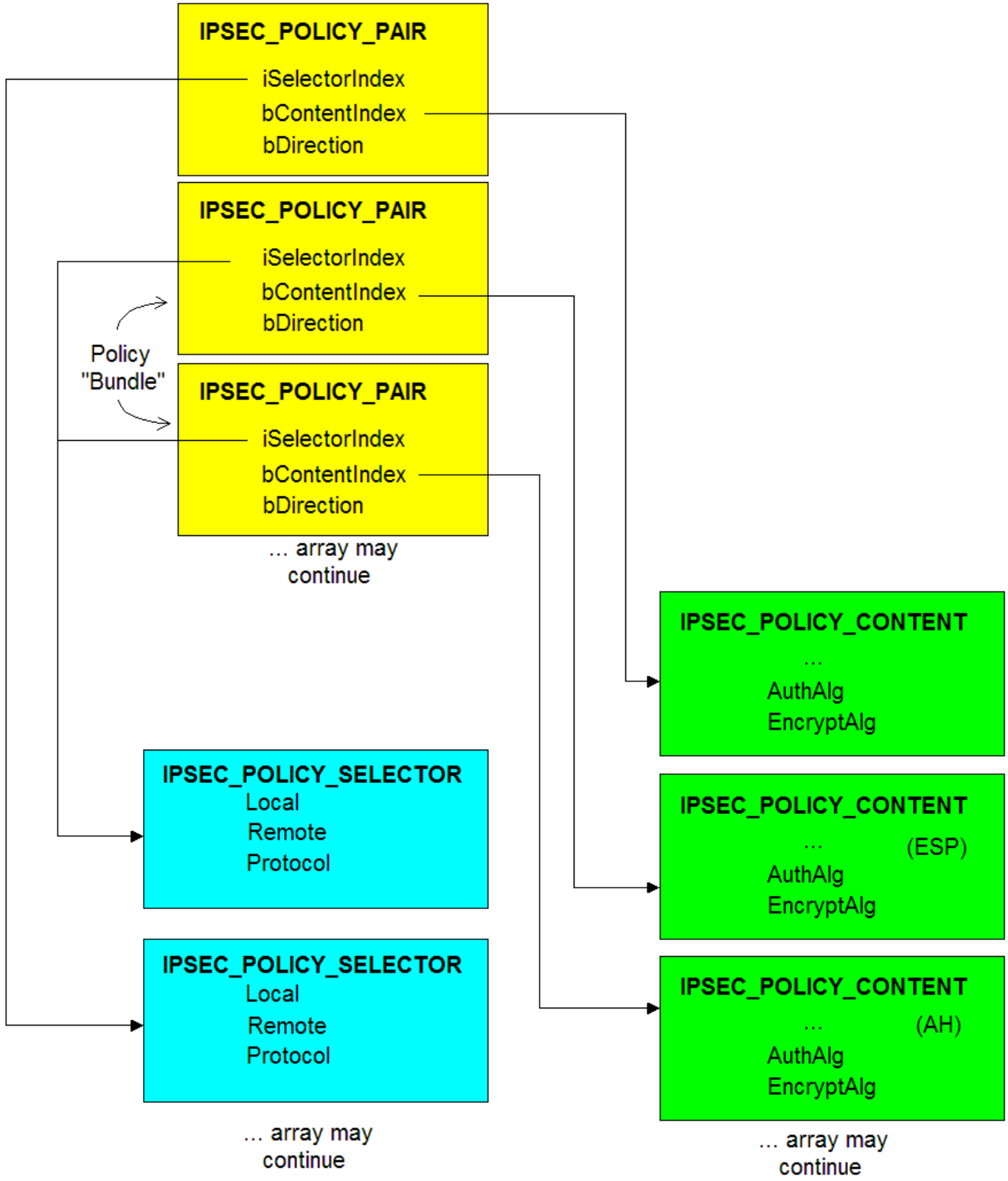
IPSEC_INBOUND: This policy will be applied to inbound IP traffic that matches the selection criteria.

IPSEC_OUTBOUND: This policy will be applied to outbound IP traffic that matches the selection criteria.

IPSEC_BOTH_DIRECTION: This policy will be applied to IP traffic in either direction that matches the selection criteria.

The relationships between the IP security policy data structures provided by the tIPSEC_POLICY_PAIR are illustrated in the diagram below. Policy bundles are formed where consecutive tIPSEC_POLICY_PAIR contain the same iSelectorIndex values. A policy bundle would be used, for example, when both AH and ESP protocols are to be applied. The protocol transform indicated by the first referenced **tIPSEC_POLICY_CONTENT** in the bundle is applied to outgoing packets first (inner header), followed by the protocol headers for the subsequent transforms in the bundle.

Note that when both ESP and AH protocols are used, the ESP is normally the inner protocol (listed first in the **tIPSEC_POLICY_CONTENT** array) and the AH protocol is the outer header (listed after ESP). This normal ordering of headers is a requirement for interfacing to Microsoft Windows (unless you know of someway to rig Windows otherwise).




**Related Topics**

API function **IPsec_Set_Policy()** - Install IPsec policies
**tIPSEC_POLICY_CONTENT** - Data structure
**tIPSEC_POLICY_SELECTOR** - Data structure

**Supported since or modified in @CHIP-RTOS version**

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

This data structure is used to select IP security policy based on IP packet address information.

```
typedef struct sIPSEC_POLICY_SELECTOR
{
    struct sIPSEC_ADDR    Local ;
    struct sIPSEC_ADDR    Remote ;
    unsigned char         Protocol ;
    unsigned char         _pad_ ;       // reserved (set to 0)

} tIPSEC_POLICY_SELECTOR ;
```

### Comments

`Local`
This **tIPSEC_ADDR**type structure specifies the IP information for the local side, to be used for policy selection. A single IP address, a range of IP addresses or a network and netmask can be specified here.

`Remote`
This **tIPSEC_ADDR**type structure specifies the IP information for the remote side. A single IP address, a range of IP addresses or a network and netmask can be specified here.

`Protocol`
Optionally, the IP protocol can be used for policy selection. Set this member to `IPSEC_WILD_PROTOCOL`to ignore IP protocol during policy selection.

### Related Topics

API function **IPsec_Set_Policy()** - Install IPsec policies
**tIPSEC_ADDR** - Data structure

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## tIPSEC_TKE_PHASE1

This data structure is used to specify a transform to be used in IKE's phase 1 proposals.

```
typedef struct sIPSEC_TKE_PHASE1
{
    unsigned int    eEncryptAlg ;
    unsigned int    eHashAlg ;
    unsigned int    eAuthMethod ;
    unsigned int    iDH_Group ;
    unsigned long   lLifeKbytes ;
    unsigned long   lLifeSeconds ;

} tIPSEC_TKE_PHASE1 ;
```

### Comments

This data structure is used by the **IPsec_IKE_Phase1_Set()** API.

`eEncryptAlg`
This enumerator specifies the encryption algorithm to be used. The options available here are:

`IPSEC_IKE_DES_CBC`= 1: Selects DES with Cipher Block Chaining.
`IPSEC_IKE_3DES_CBC`= 5: Selects 3DES with Cipher Block Chaining.
`IPSEC_IKE_AES_CBC`= 7: Selects AES ("Advanced Encryption Standard" per RFC 3602) with Cipher Block Chaining.

`eHashAlg`
This enumerator specifies the hash algorithm to be used. The options available here are:

`IPSEC_IKE_MD5`= 1: Selects MD5.
`IPSEC_IKE_SHA1`= 2: Selects SHA1.

`eAuthMethod`
This enumerator specifies the authentication method to be used. The options available here are:

`IPSEC_IKE_PRESHARED_KEY`= 1
`IPSEC_IKE_RSA_SIG`= 3

`iDH_Group`
The Diffie-Hellman group to use. Valid values are 1, 2, 5 or 14.

`lLifeKbytes`
Lifetime of Security Association (SA) in kilo-bytes. Valid values are between 10 and 2097152 (=0x200000), inclusive. Values outside this range will replaced

by the default value of 32768 kByte inside the IPsec_IKE_Phase1_Set()call. No error code is returned to indicate this out of bounds parameter.

`lLifeSeconds`
Lifetime of SA in seconds. Valid values are between 240 to 63072000 seconds (=2 years), inclusive. Values outside this range are replaced by the default value 86400 seconds (= 1 day) inside the IPsec_IKE_Phase1_Set()call. No error code is returned to indicate this out of bounds parameter.

### Related Topics

API function IPsec_IKE_Phase1_Set() - Add transform

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## tIPSEC_TUNNEL

This data structure is used to specify the IP addresses of the end points used for IP Security tunnel mode. These may be either IPv4 or IPv6.

```
typedef struct sIPSEC_TUNNEL
{
    // All IP addresses here are in network byte order.
    // Length:  IPv4 - 4 bytes (trailing 12 ignored)
    //          IPv6 - 16 bytes
    unsigned char FamilyLocal ;  // AF_INET (IPv4) or AF_INET6 (IPv6)
    unsigned char IpLocal[16] ;  // IP address of our side of tunnel.

    unsigned char FamilyRemote ; // AF_INET (IPv4) or AF_INET6 (IPv6)
    unsigned char IpRemote[16] ; // IP address of other end of tunnel.

} tIPSEC_TUNNEL ;
```

### Comments

The tIPSEC_TUNNEL structure is used to specify the host end points for an IP Security Tunnel. The local side of the tunnel must be the IPC@CHIP®. The `USE_OUR_IP` flag bit in the `FamilyLocal` member is recommended for this purpose.

Note that the BeckIPSec policy editor tool may OR in a 0x40 bit into either the `FamilyLocal` or `FamilyRemote` bytes here if the respective IP address was entered by domain name. In this case the domain name will be listed in the ASCII string bucket appended to the end of the policy file. This extra bit flag will be ignored by the system.

`FamilyLocal`

Set to either `AF_INET`for IPv4 or `AF_INET6`for IPv6. Also, the `USE_OUR_IP`flag can be OR'ed in here with either of the family indicators to signal that the the IPC@CHIP®'s own IP address should be used in place of `IpLocal`. This designation is recommended here.

`IpLocal`
This is either a four byte IPv4 or 16 byte IPv6 address, in network byte order. For the IPv4 case, the trailing 12 bytes are ignored. If the `USE_OUR_IP`flag is set in the `FamilyLocal`member, then this value is ignored.

`FamilyRemote`
Set to either `AF_INET`for IPv4 or `AF_INET6`for IPv6.

`IpRemote`
This is either a four byte IPv4 or 16 byte IPv6 address, in network byte order. For the IPv4 case, the trailing 12 bytes are ignored.

### Related Topics

API function IPsec_Set_Policy() - Install IPsec policies
tIPSEC_POLICY_CONTENT - Data structure

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

End of document

---

## IPsec_Add_Preshared_Key

This function adds a preshared key for use with the specified remote IP during Internet Key Exchange (IKE).

```
int IPsec_Add_Preshared_Key(
                    const char *remote_ip,
                    const char *preshared_key) ;
```

## Parameters

*remote_ip*

Zero terminated ASCII string specifying the IP address of the remote side.

*preshared_key*

Zero terminated ASCII string specifying the preshared key.

## Return Value

**Error code**, zero on success

## Comments

The IP security must be **started** prior to using this API.

The IP address may be either IPv4 of IPv6. Domain names must be **resolved** first by the caller and are not supported by this API.

## See Also

**IPsec_Remove_Preshared_Key()** **getHostByName()**
**IPsec_Clear_Preshared_Keys()**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **IPSEC_VECTORS** for more details.

## Related Topics

**PRESHARED_KEY** definition in CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## IP Security / IKE Error Codes

All error codes here are stated in decimal. These error codes can be viewed using the IKE console command when they are present (non-zero).

### IP Security startup error codes

```
  0       Successful.
 -2       Policy file read failed.
 -3       Policy file too large, exceeded 64 Kbyte limit.
 -4       Memory allocation failed for policy file read.
 -5       Invalid policy file read contents.
202       Policy file open failed, file not found.
222 up to 999   Some internal error has occurred.
1222      Invalid policy file contents.
1234      Policy file's policy index out of range.
1247      Policy from file not supported.
1255      Policy memory space allocation failed.
1260      Policy file contains a bad policy.
2255      Memory space allocation failed for preshared key from
policy file.
6222      Invalid preshared key specified in CHIP.INI.
6255      Memory space allocation failed for preshared key
specified in CHIP.INI.
```

### IKE error codes

Error codes 1 through 25 are received from peer in an information exchange type message.

```
 1        Invalid payload type.
 2        DOI not supported.
 3        Situation not supported.
 4        Invalid cookie.
 5        Invalid major version.
 6        Invalid minor version.
 7        Invalid exchange type.
 8        Invalid flags.
 9        Invalid message ID.
10        Invalid protocol ID.
11        Invalid SPI.
12        Invalid transform ID.
13        Attributes not supported.
14        No proposal chosen (all rejected).
15        Bad proposal syntax.
```

```
   16      Payload malformed.
   17      Invalid key info.
   18      Invalid ID information.
   19      Invalid certificate encoding.
   20      Invalid certificate.
   21      Bad certificate request.
   22      Invalid certification authority.
   23      Invalid hash info.
   24      Authentication failed.
   25      Invalid signature.

  100      Message from peer too large for receiver buffer.
  101      Received clear text message, expected encrypted.
  102      Received encrypted message, expected clear text.
  103      Invalid message received (length error).
  104      DH group not available for use in aggressive mode.
  105      Message decryption failed.
  106      Policy with void content.
  110      Received invalid phase 1 SA header.
  111      No acceptable proposal found in phase 1 SA header.
  112      Shared secret key computation failed.
  113      Expected KE header not present in received message.
  114      Expected NOUNCE header not present in received
message.
115      NOUNCE header length error.

  116      Diffie-Hellman key pair generation failed.
  117      Preshared key specified by peer not found.
  118      Unexpected header in message.
  120      Requested authentication method not supported.
  121      Install of peer certificate received into database
failed.
  122      Received message invalid (too many headers).
  123      IKE_LOCALCERT certificate missing.
  124      Requested CA certificate not found in database.
  125      Phase 1 authorization method does not fit local
certificate.
  130      No phase 1 SA proposal accepted by responder.
  131      Internal error in phase 1 SA handling.
  133      PKI certificate output message build failed.
  134      PKI signature output header build failed.
  140      Expected phase 1 ID header not present in received
message.
  141      Invalid phase 1 ID header length received.
  142      Phase 1 ID type specified by peer not supported.
  143      Phase 1 ID port of protocol incorrect.
  150      Expected phase 1 signature header not present.
  151      CA certificate needed to verify peer ID not found.
  152      Phase 1 hash computation for signature verification
failed.
  153      Phase 1 signature verification failed.
  155      Expected phase 1 hash header not present.
  156      Phase 1 hash header lengthh invalid.
  157      Phase 1 hash compare failed for received hash.

  160      Quick Mode Initiator: No valid policy data available.
  161      Quick Mode Responder: No policy data found.
  162      Quick Mode: Internal error generating SA header.
  163      Quick Mode: Diffie-Hellman group invalid.
  170      Quick Mode: Expected ID header not present.
  171      Quick Mode: ID header length invalid.
  173      Quick Mode: ID protocol mismatch.
  174      Quick Mode: ID type not supported.
```

```
180      Quick Mode: Expected SA header not present.
181      Quick Mode: Unexpeced header.
182      Quick Mode: Hash header length invalid.
185      Quick Mode: Invalid SA header from peer.
186      Quick Mode: No acceptable SA proposal found.
187      Quick Mode: KE header either missing or unexpected.
189      Quick Mode: Malformed CONNECTED notification.
255      Memory allocation failed.
```

## Related Topics

**IKE** console command
IP Security **API**

---

End of document

---

## IPsec_IKE_Phase1_Set

Specify a transform for IKE's phase 1 proposal.

```
int IPsec_IKE_Phase1_Set(int index,
                  const tIPSEC_TKE_PHASE1 *param) ;
```

## Parameters

*index*

Index 0 or 1 selects which of the two phase 1 IKE transforms is to be set.

*param*

Pointer to a **tIPSEC_TKE_PHASE1** data structure used to specify the transform's parameters. A copy of this data is made inside this API, so the caller's image need not persist beyond the return from this function.

## Return Value

**error code**, zero on success.

## Comments

This function adds a user specified phase 1 transform to IKE's phase 1 proposal. The user is able to configure up to two such transforms. The most preferred has index 0, the less preferred has index 1. If the user configures either of these, the default transform will not be used.

When neither of these optional phase1 proposals is set, the default transform will be used in the phase 1 proposal. This default transform has the following settings:

Encryption Algorithm: 3DES CBC
Hash Algorithm: SHA1
Diffie-Hellman Group: 2
SA Lifetime seconds: 86400 seconds (=1 day)
SA Lifetime kBytes: 0x10000 kByte (=64 MByte)

The default authentication method is **preshared key** unless a local **certificate** has been specified in the CHIP.INI file.

If this API is used to set both of the phase 1 transforms available for user control, then the authentication method specified in these two transforms must be set the same. Otherwise unpredictable operation may result.

## See Also

IPsec_IKE_Phase1_Delete()

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function IPSEC_VECTORS for more details.

## Related Topics

IKE_PH1_AGGRESSIVE definition in CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## IPsec_Set_Option

Set options controlling IPsec/IKE behavior.

```
int IPsec_Set_Option(unsigned int option,
                     unsigned int value) ;
```

## Parameters

*option*

An enumerator that specifies which option to set.

*value*

The options value to be set.

## Return Value

0: Success
-1: Failure, invalid option or value

## Comments

This function allows the IP Security and Internet Key Exchange (IKE) protocol options to be adjusted at run-time. The IP Security function must have been started either from the **CHIP.INI** or **IPsec_Start()** API before using this API.

For all options here which have a Boolean sense, a value 1 enables the option and 0 disables the option.

The options available are:

`IPSEC_OPT_ANTIREPLAY`= 1 Specifies if anti-replay is used or not. By default anti-replay is used (value = 1).

`IPSEC_OPT_PFSKEY`= 2 Specifies if Perfect Forward Secrecy (PFS) is used of not. By default PFS is used (value = 1).

`IPSEC_OPT_AGGRESS`= 3 Specifies if aggressive mode is used in IKE phase 1. By default this option is not selected (value = 0).

`IPSEC_OPT_AGGRESSDHGROUP`= 4 Specifies which Diffie-Hellman group is going to be used in aggressive mode. Value must be 1, 2, 5 or 14. The default is group 2 (using 1024 bit prime number).

`IPSEC_OPT_INITCONTACT`= 5 Specifies if Initial Contact message is to be sent to the peer or not. Value 1 will enable IKE to send Initial contact upon finishing IKE phase 1 negotiation. Value

0 will disable it. By default this option is enabled.

`IPSEC_OPT_ICMPBYPASS= 9` Specifies if the policy check for incoming ICMP packets is bypassed or not. Value 1 will bypass the incoming policy check for incoming ICMP packets. By default this option is selected (value = 1).

`IPSEC_OPT_ICMPSRCCHKBYPASS= 10` Specifies if ICMP source address should be checked against IPsec policy or not. Value 1 will bypass this check. By default this option is not selected (value = 0).
`IPSEC_OPT_ICMP6NDMLDBYPASS= 11` Specifies if ICMPv6 packets bypass IPsec policy or not. Value 1 will bypass IPsec. By default this option is selected (value = 1).

`IPSEC_OPT_NESTIKE_BYPASS= 12` Specifies if nested IKE packets are to be checked against IPsec policy or not. Value 1 will bypass all IKE packets. Value 0 will force all IKE packets to go through IPsec policy checking. By default this option is not selected (value = 0).

`IPSEC_OPT_DFBIT= 13` Specifies how to set the outside IP header's *Dont Fragment*bit if a tunnel is used. Valid values are:

```
IPSEC_VAL_DFBIT_CLEAR= 0
IPSEC_VAL_DFBIT_SET= 1
IPSEC_VAL_DFBIT_COPY= 2
```

By default, the `IPSEC_VAL_DFBIT_COPY= 2` value is used.

## See Also

**IPsec_Start()**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **IPSEC_VECTORS** for more details.

## Related Topics

**IKE_PH1_AGGRESSIVE** definition in CHIP.INI file
**IKE_PH2_PFS** definition in CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

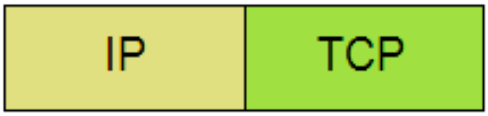
## IP Security Overview

IP Security **API**
IP Security **data structures**
IPsec **NAT-Transversal**

IP Security (IPsec) can provide data encryption and/or authentication. This is implemented at the IP level for either IPv4 or IPv6. IP Security is implemented with two additional IP header types:

- Authentication Header (AH)
- Encapsulated Security Payload (ESP)

In the discussion below the transport of the following IP message will be considered:




The use of TCP as the upper protocol here is taken only for illustration. Any other upper layer protocol would work in the same manner.

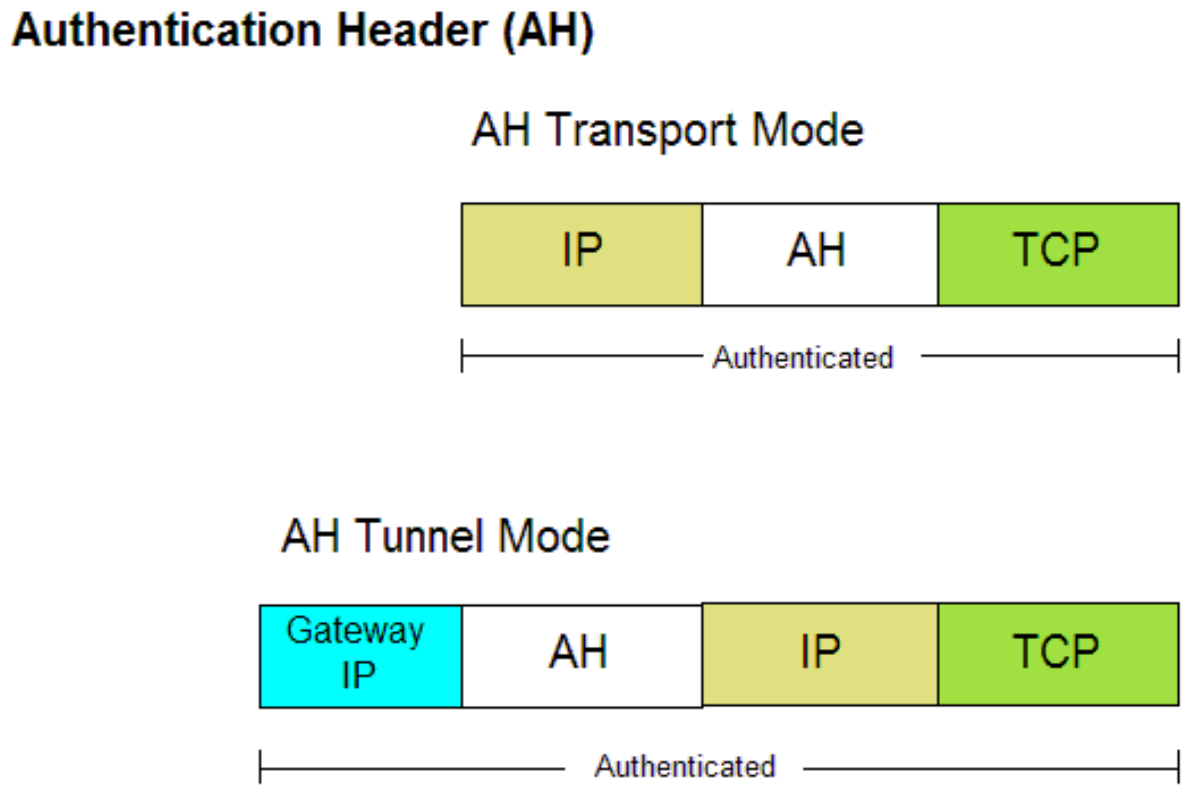
IP Security offers a choice of two different encapsulation modes:

- **Transport Mode** - In this mode the IP packet travels directly between two peers.
- **Tunnel Mode** - In this mode the IP packet is re-addressed to first be passed to a specified remote security gateway computer. The IPC@CHIP® computer plays the role as the local gateway computer at which the tunnel begins. The gateway computers will use the original IP header that is nested inside the tunneled IP packet to forward this packet to it's final destination in plain text form, after performing the necessary decryption and authentication.

## Authentication Header

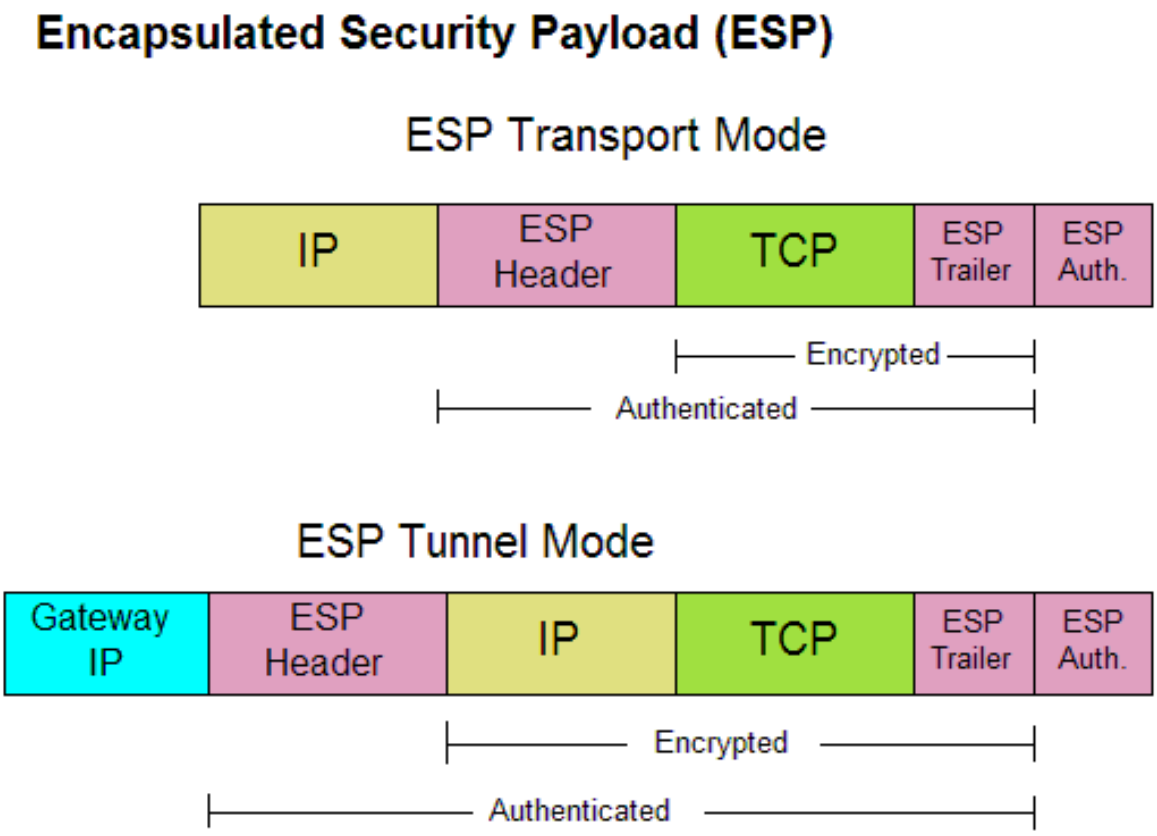
The Authentication Header provides a hash value over the encapsulated message and the IP header. This hash value can be used by the receiving side to verify that the message has originated from the intended source and that no bit of the message has been altered in route. The resulting IP packet will look as follows, depending on whether tunnel or transport mode is selected.

The AH protocol cannot be used when a **NAT** device is in the path between IPsec peers. The fundamental problem here is that NAT devices make changes to the IP header, which invalidates the AH hash value.



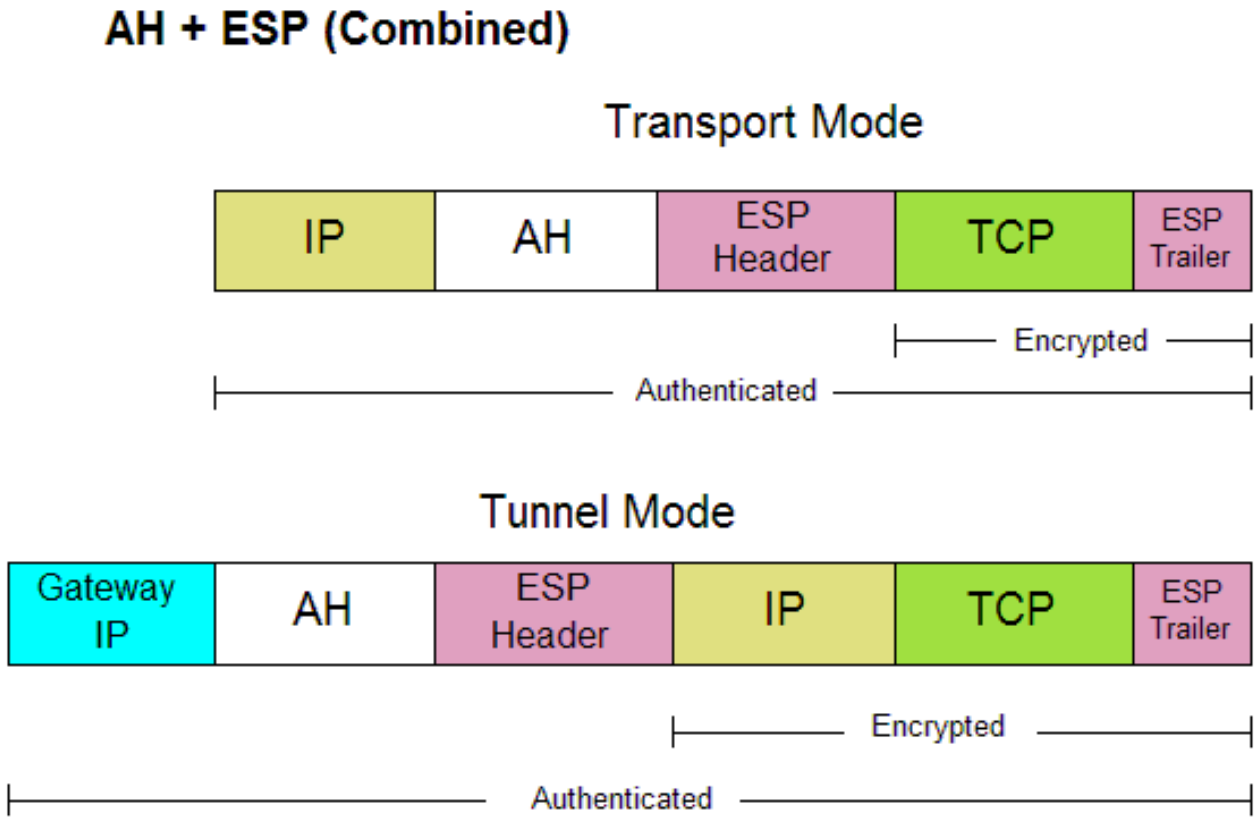

## Encapsulated Security Payload

The Encapsulated Security Payload protocol provides encryption of the IP message. It may also optionally be used to append a hash value for message authentication. However, note that the ESP authentication does not cover the IP header (see diagrams below) as does the heavier duty AH type authentication.




## ESP and AH Combined

The illustration below shows the conventional way of combining ESP and AH. (The specifications for these protocol would allow other variations as well.) When ESP and AH are both applied to IP messages, the ESP is applied first as the inner protocol. This is then wrapped with the AH header.




For tunnel mode, the ESP protocol is used in tunnel mode while the AH operates in transport mode. Note that

the optional authentication offered by ESP is not used in this example where both ESP and AH are used. The AH should provide adequate authentication.

## IPsec NAT-Transversal

In IPv4 networks, Network Address Translator (NAT) functionality is commonly found inside routers connecting to the Internet. The AH protocol cannot be used when a NAT device is in the path between IPsec peers. ESP protocol can be made to work across NAT devices by encapsulating the ESP data inside UDP datagram.

The IPC@CHIP® implementation provides negotiation of IPsec NAT-Transversal per RFC-3947 and ESP protocol encapsulation with UDP per RFC-3948. The "draft 2" of the RFC-3947 ("draft-ietf-ipsec-nat-t-ike-02") negotiation is also supported. The "draft 2" support enables operation with current (year 2007) Microsoft Windows XP NAT-Transversal implementations for ESP transport mode. ESP tunnel mode operation between Microsoft Windows XP and the IPC@CHIP® has not been achieved. However, ESP tunnel mode operation between @Chip-RTOS devices across a NAT will work.

By default, this NAT-transversal capability is enabled. It can be deselected with the CHIP.INI [IPSEC] **NAT** entry.

---

End of document

---




# @CHIP-RTOS - RTOS API



## RTOS API

RTOS API Here is the documentation for the RTOS API. This interface provides access to the RTOS of the IPC@CHIP®.
For a general introduction about multitasking with the @CHIP-RTOS see **Multitasking introduction**. Please note that we cannot explain in detail all principles of a multitasking system in this document.
A good book for understanding the architecture of Real-Time-Kernels is "MicroC/OS" from Jean.J.Labrosse.

For some useful comments see also **Programming notes**.

### Topics

RTOS API **C-Library**
RTOS API **Overview**
RTOS API **Error Codes**
RTOS API **Developer**Notes
RTOS API **Examples**Available
RTOS API **Data Structures**
IPC@CHIP®System **Tasks**

### Task control functions

- **0x00: _RTX_SLEEP_TIME, Sleep for a specified time**
- **0x01: _RTX_TASK_CREATE, Create and start a task**
- **0x11: _RTX_TASK_CREATE_WITHOUT_RUN, Create a task**
- **0x02: _RTX_TASK_KILL, Stop and kill specified task.**
- **0x03: _RTX_TASK_DELETE, Remove a task from the system**
- **0x04: _RTX_GET_TASKID, Get ID of the current task**
- **0x05: _RTX_SLEEP_REQ, Sleep until wake request**
- **0x06: _RTX_WAKEUP_TASK, Wake up a task**
- **0x07: _RTX_END_EXEC, End execution of task**
- **0x08: _RTX_CHANGE_PRIO, Change priority of a task**
- **0x09: _RTX_ACCESS_FILESYSTEM, Reserve file access context**
- **0x0A: _RTX_GET_TASK_STATE, Get state of a task**
- **0x0B: _RTX_GET_TASK_LIST, Get list of tasks**
- **0x0C: _RTX_START_TASK_MONITOR, Enable task monitoring**
- **0x0D: _RTX_STOP_TASK_MONITOR, Disable task monitoring**
- **0x0E: _RTX_SUSPEND_TASK, Suspend a task**
- **0x0F: _RTX_RESUME_TASK, Resume a task**
- **0x10: _RTX_RESTART_TASK, Start task**

- 0x12: _RTX_GET_TASK_STATE_EXT, Get task state
- 0x13: _RTX_COUNT_RESOURCES, Check resources available.
- 0x1D: _RTX_SLEEP_FINE, Sleep for RTI count
- 0x20: _RTX_DISABLE_TASK_SCHEDULING, Task Lock
- 0x21: _RTX_ENABLE_TASK_SCHEDULING, Release Task Lock

### Semaphore services

- 0x14: _RTX_CREATE_SEM, Create a semaphore
- 0x15: _RTX_DELETE_SEM, Delete a semaphore
- 0x16: _RTX_FREE_RES, Free a resource semaphore
- 0x17: _RTX_GET_SEM, Get counting semaphore (no wait)
- 0x18: _RTX_RELEASE_SEM, Release a resource semaphore
- 0x19: _RTX_RESERVE_RES, Get use of a resource semaphore
- 0x1A: _RTX_SIGNAL_SEM, Signal a counting semaphore
- 0x1B: _RTX_WAIT_SEM, Wait on a counting semaphore
- 0x1C: _RTX_FIND_SEM, Find semaphore by name
- 0x1E: _RTX_WAIT_SEM_PRIO, Wait on a a semaphore

### Time/Date functions

- 0x28: _RTX_GET_TIMEDATE, Get system time and date
- 0x29: _RTX_SET_TIMEDATE, Set system time and date
- 0x2A: _RTX_GET_TICKS, Get tick count of system clock
- 0x2B: _RTX_RTI_COUNT, Get real-time interrupt count.

### Timer procedures

- 0x2F: _RTX_TIMER_FINE, Install a fine precision timer procedure
- 0x30: _RTX_INSTALL_TIMER, Install a timer procedure
- 0x31: _RTX_REMOVE_TIMER, Remove a timer procedure
- 0x32: _RTX_START_TIMER, Start periodic timer procedure
- 0x33: _RTX_STOP_TIMER, Stop execution of a timer procedure
- 0x34: _RTX_TIMER_DELAY, Set timer delay
- 0x35: _RTX_SET_RTI_HOOK, Install Real-Time Interrupt hook

### Event group manager

- 0x40: _RTX_CREATE_EVENTGROUP, Create an event group
- 0x41: _RTX_DELETE_EVENTGROUP, Delete an event group
- 0x42: _RTX_SIGNAL_EVENTS, Signal event(s) in a group
- 0x43: _RTX_WAIT_EVENTS, Wait for events in a group
- 0x44: _RTX_GET_EVENTGROUP_STATE, Read the event states
- 0x45: _RTX_GET_EVENT_FLAGS, Get the saved event flags
- 0x46: _RTX_FIND_EVENTGROUP, Find an event group

### Message exchange manager

- 0x50: _RTX_CREATE_MSG, Create a Message Exchange
- 0x51: _RTX_DELETE_MSG, Delete a Message Exchange

- 0x52: _RTX_SEND_MSG, Send message
- 0x53: _RTX_GET_MSG, Poll Message Exchange
- 0x54: _RTX_WAIT_MSG, Wait for a message
- 0x55: _RTX_FIND_MSG, Find a Message Exchange

## RTX Interrupt 0xAD Services

The RTOS API uses interrupt 0xAD with a service selector in the high order byte of the AX register (AH) as specified here:

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 use with 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, Sleep for a specified time

### Parameters

*AH*

0 (=RTX_SLEEP_TIME)

*BX*

Sleep time in milliseconds. A value of zero is rounded up to 1.

### Return Value

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

### Comments

The **RTX_WAKEUP_TASK** API (service 0x06) can wake up a sleeping task before its sleep timer has expired. In this case this API returns with error code -5 in AX.

The system maintains a "*Task Wakeup Pending* " flag for each task. This flag is set when **RTX_WAKEUP_TASK** is called for a task which is not currently awaiting wakeup (i.e. when bit 7 in **Task State** is zero). In the case where "*Task Wakeup Pending* " flag had been set, this RTX_SLEEP_TIME API clears this flag and returns immediately with error code -5 in AX.

A sleep call with BX parameter 1 millisecond sleeps for less than or equal to one millisecond. If a user needs a minimum sleep time of 1 millisecond they must call RTX_SLEEP_TIME with value 2 in BX. In general with BX sleep time specified to be N milliseconds, the resulting sleep time will range from N-1

milliseconds up to N milliseconds (inclusive).

## Beck C-Library Usage

RTX_Sleep_Time()

## Related Topics

RTOS Task Control Services
RTX_SLEEP_REQ Sleep until wake up call
RTX_WAKEUP_TASK Wake up a task
RTX_SLEEP_FINE Sleep for specified RTI ticks.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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 TASK_PARAM_DEF type data structure. (The deprecated TaskDefBlock structure may also be used here.)

## 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 TASK_PARAM_DEF 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). The alternate API RTX_TASK_CREATE_WITHOUT_RUN can be used if it is not desired that the task be free to run immediately on creation.

## Beck C-Library Usage

RTX_Create_Task()

RTX_Param_Task()

## Related Topics

IPC@CHIP® System Tasks
RTOS Task Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x11: RTX_TASK_CREATE_WITHOUT_RUN, Create a task

## Parameters

*AH*
0x11 (= RTX_TASK_CREATE_WITHOUT_RUN)

*BX:SI*
Pointer to 16 bit storage for the taskID, allocated by the caller

*ES:DI*
Pointer to a TASK_PARAM_DEF type data structure. (The deprecated TaskDefBlock structure may also be used here.)

## 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 TASK_PARAM_DEF structure prior to making this call.

Unlike the alternative RTX_TASK_CREATE API, this API call does not start the new task. The new task can be started with a RTX_RESTART_TASK call.

## Beck C-Library Usage

RTX_Create_Task_Without_Run()
RTX_Param_Task_Without_Run()

## Related Topics

IPC@CHIP® System Tasks
RTOS Task Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x02: RTX_TASK_KILL, Stop and kill specified task.

Terminate a specified task, but do not remove it from system.

## 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

## Comments

You should not kill a task which is waiting for a semaphore, an event in an event group, or a message from a Message Exchange. Failure to observe this restriction can lead to unpredictable results.

This function does not remove the task from the system. The task can be restarted by calling **RTX_RESTART_TASK** API function.

## Beck C-Library Usage

**RTX_Kill_Task()**

## Related Topics

RTOS **Task** Control Services
**RTX_TASK_DELETE** Remove task from system
**RTX_END_EXEC** Task terminates itself

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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 a task which is waiting for a semaphore, an event in an event group, or a message from a Message Exchange. Failure to observe this restriction can lead to unpredictable results.

After making this call, the taskID is no longer valid. A task can delete itself.

CAUTION:

Any memory allocated from the system by the task which is being deleted here will be released.

### Beck C-Library Usage

**RTX_Delete_Task()**

### Related Topics

RTOS **Task** Control Services
**RTX_TASK_KILL** Terminate task execution

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x04: RTX_GET_TASKID, Get ID of the current task

Returns ID of the calling task.

### Parameters

*AH*

0x04 (= RTX_GET_TASKID)

## Return Value

DX=0 (success always) AX: contains the TaskID

## Beck C-Library Usage

**RTX_Get_TaskID()**

## Related Topics

RTOS **Task** Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x05: RTX_SLEEP_REQ, Sleep until wake request

The calling task will be suspended until some other task issues a **RTX_WAKEUP_TASK** call to awake this calling task.

## Parameters

*AH*
0x05 (= RTX_SLEEP_REQ)

## Return Value

DX=0 (always success) AX: 0

## Comments

The system maintains a "*Task Wakeup Pending* " flag for each task. This flag is set when **RTX_WAKEUP_TASK** is called for a task which is not currently awaiting wakeup (i.e. when bit 7 in **Task State** is zero). In the case where "*Task Wakeup Pending* " flag had been set, this RTX_SLEEP_REQ API clears this flag and returns immediately.

## Beck C-Library Usage

**RTX_Sleep_Request()**

## Related Topics

RTOS **Task** Control Services
**RTX_WAKEUP_TASK** Wake up a task

RTX_SLEEP_TIME Timed sleep

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x06: RTX_WAKEUP_TASK, Wake up a task

To wake up a task known to be waiting because of a RTX_SLEEP_REQ or RTX_SLEEP_TIME call.

## Parameters

*AH*
0x06 (= RTX_WAKEUP_TASK)

*BX*
taskID

## Return Value

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

## Comments

An immediate task switch will occur if the task being wakened is of higher priority than the current task.

If this API is called for a task which is not currently awaiting wakeup, error code -6 is returned. In this case a wakeup is left pending such that when the specified task eventually makes a call to RTX_SLEEP_REQ or RTX_SLEEP_TIME API, it will react (once) to this pending wakeup and return immediately without a sleep period.

## Beck C-Library Usage

RTX_Wakeup()

## Related Topics

RTOS Task Control Services
RTX_SLEEP_REQ Sleep until wake up call
RTX_SLEEP_TIME Timed sleep

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAD service 0x07: RTX_END_EXEC, End execution of task

This call terminates the calling task.

### Parameters

*AH*
0x07 (= RTX_END_EXEC)

### Return Value

There is no return from this function

### Comments

This call is equivalent to returning to the system from a task's main procedure.

The task can later be restarted with the **RTX_RESTART_TASK** API.

### Beck C-Library Usage

**RTX_End_Execution()**

### Related Topics

RTOS **Task** Control Services
**RTX_TASK_KILL** Kill specified task

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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 or zero for current task

*CX*
Priority, range 2 to 127 inclusive (2 is highest priority).

Or set to PRIORITY_READ_ONLY flag (=0x5ABC) to indicate that only a read out of the existing task priority is desired.

## Return Value

DX =0 success
AX: 0,
CX: existing task priority (before any change)
DX!=0 failure AX: contains **error** code

## Comments

The existing task priority is reported in the CX register. This is what the task priority was before any change made by this call. This function can be used to read out the existing task priority without modifying the priority by setting the CX register to the special flag setting 0x5ABC.

A task can read/modify its own priority by setting the taskID in BX register to zero. The **AMXK** kernel task's priority cannot be changed.

An out of range priority value (CX) will be limited to range 2..127 inside this function. (CX = 0x5ABC is an exception for which no priority change is performed.)

**Note:**

Internally all tasks have a unique priority. When a task is **created** or its priority is changed, that task is given a lower internal task priority than any other task in the system with the same user task priority.

## Beck C-Library Usage

**RTX_Change_TaskPrio()**
**RTX_Task_Priority()**

## Related Topics

RTOS **Task** Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x09: RTX_ACCESS_FILESYSTEM, Reserve file access context

Reserve file access context space for the calling task.

## Parameters

*AH*
0x09 (= RTX_ACCESS_FILESYSTEM)

## Return Value

DX =0, AX =0: Success
DX = -1, AX = -1: Failure (Too many tasks with file access)

## Comments

This API is deprecated. (However, it may still have some diagnostic value to determine why file accesses are failing or could be used to acquire the file access context space ahead of time to assure the task's future file accesses will succeed.)

It is no longer necessary that a task use this API to gain access to the file system. A task is automatically granted file system access when the file system API is used, provided that the system has not ran out of file access context space.

The file system supports registration of up to 15 tasks at a time. Each DOS program's main task is registered with the file system regardless of whether or not that program uses the file system. A task's reservation on the file system access context space remains in effect until that task is deleted from the system. The DOS program main tasks are deleted when the program terminates (or **terminates and stays resident**).

Once a successful call to this API has been made, any further calls perform no operation (at most one context slot is assigned per task).

If this API returns a failure indication, then the number of tasks registered with the file system has reached the limit and this calling task is consequently not being granted access to the file system.

## Beck C-Library Usage

**RTX_Access_Filesystem()**

## Related Topics

RTOS **Task** Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x0A: RTX_GET_TASK_STATE, Get state of a task

Get state information for a specified 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. This need not be a null terminated, but must be four bytes.

*DS:SI*

Output parameter: Pointer to `Task_StateData` type **structure** allocated by the user to be filled by this API

## Return Value

On Success:

DX = 0
AX = taskID
`Task_StateData`**structure**at [DS:SI] contains the current task state data

Failure:

DX = -1
IF AX is zero THEN
"Task **Monitoring**is not enabled".
ELSE
"Specified task not found".
ENDIF

## Comments

Task monitoring mode must first be enabled in order for this API to work.

The alternative function, **RTX_GET_TASK_STATE_EXT**, can be used without starting the Task Monitor, but offers less information about the task.

## Beck C-Library Usage

**RTX_Get_Task_State()**

## Related Topics

`Task_StateData` **structure** definition
RTOS **Task** Control Services
**Start** Task Monitor API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x0B: RTX_GET_TASK_LIST, Get list of tasks

Get list of current tasks in the system.

## Parameters

*AH*

0x0B (= RTX_GET_TASK_LIST)

*ES:DI*

Output Parameter: Pointer to array of `TaskList` **type** structures allocated by user.

*CX*

Length of the list, number of data structures in user's array at [ES:DI].

## Return Value

DX=0
BX = number of tasks listed in [ES:DI] output array.

## Comments

For a full report, the caller must allocate sufficient buffer space at [ES:DI] to allow all tasks to be reported including those **created** by the system. At most, CX tasks will be reported.

## Beck C-Library Usage

**RTX_Get_Task_List()**

## Related Topics

`TaskList` **structure** definition
RTOS **Task** Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x0C: RTX_START_TASK_MONITOR, Enable task monitoring

Installs an alternate 1000 Hz timer interrupt which monitors task execution.

## 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 (or most recently if system is idle). **Data** collected by this timing function provides a coarse indication of which tasks are occupying the CPU.

Note that the Task Monitor places a considerable load on the system.

## Beck C-Library Usage

RTX_Start_Task_Monitor()

## Related Topics

Get Task state
Stop Task Monitor API call
RTOS Task Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x0D: RTX_STOP_TASK_MONITOR, Disable task monitoring

Remove special 1000 Hz timer interrupt service used by Task Monitor function and restore original handler.

## Parameters

*AH*
0x0D (= RTX_STOP_TASK_MONITOR)

## Return Value

DX=0, AX=0

## Comments

This function installs the system's normal timer 0x13 interrupt handler. This action is performed irrespective of whether or not the alternate 1000 Hz Task Monitor handler had been installed (RTX_START_TASK_MONITOR API).

## Beck C-Library Usage

RTX_Stop_Task_Monitor()

## Related Topics

RTOS Task Control Services
Start Task Monitor API call

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x0E: RTX_SUSPEND_TASK, Suspend a task

Suspend the execution of a specified task until **RTX_RESUME_TASK** is called 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

## Comments

Note that the @CHIP-RTOS implementation maintains a separate Boolean representing "Suspended" for each task. Consequently, to suspend a task which is already waiting for some other **reason** (Trigger, Semaphore, Event Group, Message, or Sleep) means that the task will remain inactive after the other wait condition is released (e.g. after the task is granted a semaphore).

## Beck C-Library Usage

**RTX_Suspend_Task()**

## Related Topics

RTOS **Task** Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x0F: RTX_RESUME_TASK, Resume a task

Re-enables 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

## Comments

This can result in an immediate task switch if the suspended task is higher **priority** than the calling task.

## Beck C-Library Usage

**RTX_Resume_Task()**

## Related Topics

RTOS **Task** Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x10: RTX_RESTART_TASK, Start task

Start execution of a specified task which was in the "Trigger Wait" **state**.

## Parameters

*AH*
0x10 (= RTX_RESTART_TASK)

*BX*
taskID

## Return Value

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

## Comments

A task is in the "Trigger Wait" state either before it has been started for the first time after being created by the **RTX_TASK_CREATE_WITHOUT_RUN** API, or after termination.

A task termination results from either returning to the system from a task's entry procedure or due to API calls **RTX_TASK_KILL** or **RTX_END_EXEC**.

## Beck C-Library Usage

**RTX_Restart_Task()**

## Related Topics

RTOS **Task** Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x12: RTX_GET_TASK_STATE_EXT, Get task state

Get state of specified task without task monitoring mode active.

## 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

Success:

DX = 0
AX = taskID
BX = task state bit **field**

Failure (task not found)

DX != 0

## Comments

The task name is not a null terminated string. It must contain four bytes (not necessarily ASCII).

The task state returned in BX is a bit **field** coded same as `taskState` member of the `Task_StateData` data structure.

## Beck C-Library Usage

**RTX_Get_Task_State_Ext()**

## Related Topics

API function **RTX_GET_TASK_STATE** - Get state of a task

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x13: RTX_COUNT_RESOURCES, Check resources available.

Read out the number of RTX resources available and optionally obtain detailed data about resources in use.

## Parameters

*AH*

0x13 (= RTX_COUNT_RESOURCES)

*ES:DI*

Two related functions:

1) ES non-zero Output Parameter: Pointer to an **RTX_COUNTS** data structure that will be filled by this API.

2) ES zero: Input Parameter: DI contains the RTX object handle for a semaphore, Event Group or Message Exchange for which the queue of tasks awaiting this object will be reported. (This supports the Beck C-Library RTX_Wait_Queue() function.)

*BX:SI*

Optional Output parameter: Pointer to a buffer which will receive objects which can be referenced with the **RTX_DETAILS** union of structure pointers. This API will report various details for each RTX object to this buffer. This parameter can be set to NULL if no detailed RTX object data is desired.

*CX*

Number of bytes in buffer at [BX:SI]. If either this value or BX is zero, then no `RTX_DETAILS` reporting is done.

*DX*

Optional output listing filter. This bit field selects which type RTX objects are listed in detailed report at [BX:SI]. Set this to zero to report all object types. When calling with ES set to zero, this parameter is required to be one of the masks `RTX_FILT_SEM` , `RTX_FILT_EVENT_GROUP` or `RTX_FILT_MSGXCHG` to identify the RTX object type whose handle is passed in DI.

## Return Value

AX < 0: Minus the number of bytes required at [BX:SI] to report task queue information for case where ES was zero and either BX was zero (NULL pointer) or CX was not large enough. This reported size can be used to allocate sufficient memory for a subsequent call.
AX >= 0: Number of **RTX_DETAILS** unions output to array at BX:SI. This object count will be limited to assure that the byte count limit value specified by CX input parameter is not exceeded. All data has been reported if this value matches the `nReport_Cnt` vallue reported in the **RTX_COUNTS** data structure (for ES != 0 case). When ES was set to 0 to read out task list waiting at a queue, a positive AX return value indicates that all the data has been reported.

## Comments

The primary purpose for this API is to allow the user to determine the number of RTX resources that are available. For this limited purpose, only the **RTX_COUNTS** data structure is required at [ES:DI] and BX or CX can be set to zero to indicate that no **RTX_DETAILS** reporting is desired.

When more detailed information about the various RTX objects in use is desired, a buffer of sufficent length must be provided by the caller at [BX:SI]. This buffer will receive a sequence of various size data structures which can be accessed with the assistance of the **RTX_DETAILS** union of pointers. The length of your buffer in bytes is indicated by the CX register contents. This API will not exceed this space.

The report filter parameter in DX allows you to focus the detailed reporting on particular types of RTX objects. This can simplify your memory management problem by reducing the size of the `RTX_DETAILS` array at BX:SI required to report all the objects of the type you are interested in. Bits set to one in this bit field inhibit the reporting of the RTX object type associated with that bit. The filter bit assignments are as follows:

0x01 - Inhibits semaphore detail reporting
0x02 - Inhibits timer detail reporting
0x04 - Inhibits Event Group detail reporting
0x08 - Inhibits Message Exchange detail reporting
0x10 - Inhibits task detail reporting

When calling with ES = 0 for wait queue information, the filter value in DX should have only the single zero bit, the ones complement of above one bit values, to indicate the type of RTX object.

One possible strategy for allocating the details buffer at [BX:SI] would be to first call this API with BX set to zero only to discover how large a buffer will be required to collect the desired detailed information. The number of bytes required in the array is reported by the API in the `nReport_Bytes` member of the **RTX_COUNTS** data structure. This reported count is influenced by what ever reporting filter you had specified in the DX register.

Inside this API, the RTX data is collected with the **task switching** locked. Interrupts are enabled after inhibiting task switching. The interrupt mask is restored to the state on entry prior to re-enabling the task switching on return. Nevertheless, a task switch can occur at the moment task switching is re-enabled here, thereby defeating any interrupt mask set by the caller.

## Beck C-Library Usage

**RTX_Count_Resources()**
**RTX_Wait_Queue()**

## Related Topics

API function **RTX_GET_TASK_STATE** - Get state of a task

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x14: RTX_CREATE_SEM, Create a semaphore

Create a resource or counting semaphore.

## Parameters

*AH*
0x14 (= RTX_CREATE_SEM)

*BX:SI*
Pointer to 16 bit storage allocated by caller where this API will output a semaphoreID

*CX*
Initial value:
Set to -1 for resource semaphore
Set in range [0 .. 32767] (inclusive) for counting semaphore

*ES:DI*
Pointer to 4 character unique name tag for the new semaphore, which need not be null terminated but must contain four bytes.

## Return Value

DX =0 success AX: 0, Location referenced by [BX:SI] contains the unique semaphoreID
DX!=0 failure AX: contains **error** code

## Comments

A resource semaphore is created by setting CX = -1. A non-negative value in CX creates a counting semaphore.

A resource semaphore is created in the free state, ready for use.

A counting semaphore is initially available the number of times specified in CX. The **RTX_SIGNAL_SEM** API increments this count and semaphore access via **RTX_GET_SEM** or **RTX_WAIT_SEM** decrements the count. A counting semaphore is not available when its count reaches zero.

## Beck C-Library Usage

**RTX_Create_Sem()**

## Related Topics

RTOS **Semaphore** Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x15: RTX_DELETE_SEM, Delete a semaphore

Removes specified semaphore from system.

### 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

### Comments

You must be certain that no other task, Interrupt Service Routine or **Timer** procedure is in any way using or about to use this semaphore. Failure to observe this restriction can lead to unpredictable faults.

### Beck C-Library Usage

**RTX_Delete_Sem()**

### Related Topics

RTOS **Semaphore** Services

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x16: RTX_FREE_RES, Free a resource semaphore

The resource semaphore's use count is set to zero which unconditionally frees the resource.

### Parameters

*AH*

0x16 (= RTX_FREE_RES)

*BX*

ID of the resource semaphore acquired from **RTX_CREATE_SEM**

## Return Value

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

## Comments

Only the task owning the resource semaphore can make this call successfully. Error code -12 is returned if the calling task does not own the semaphore (or if the semaphore is a counting semaphore type which are not owned by tasks).

This API is useful to unwind in one stroke N calls made to **RTX_RESERVE_RES**. Alternatively, the **RTX_RELEASE_SEM** API could be called N times to release the resource semaphore.

After being freed, the resource will immediately be given to the task (if any) which is waiting at the head of this resource semaphore's wait queue. An immediate task switch occurs if this waiting task is higher **priority** than the calling task that just gave up ownership of this semaphore.

## Beck C-Library Usage

**RTX_Free_Sem()**

## Related Topics

**RTX_RESERVE_RES** Reserve resource semaphore
**RTX_RELEASE_SEM** Release resource semaphore
RTOS **Semaphore** Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x17: RTX_GET_SEM, Get counting semaphore (no wait)

Attempt acquisition of a counting semaphore without waiting.

## Parameters

*AH*

0x17 (= RTX_GET_SEM)

*BX*

ID of the semaphore acquired from **RTX_CREATE_SEM**

## Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, semaphore is in use

## Comments

This call returns an error code -51, "semaphore busy", if the semaphore is not available.

This function must not be called for resource type semaphores, as this will foul up the operation of the resource semaphore.

## Beck C-Library Usage

**RTX_Get_Sem()**

## Related Topics

**RTX_WAIT_SEM** Wait for counting semaphore access
**RTX_SIGNAL_SEM** Signal counting semaphore
RTOS **Semaphore** Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x18: RTX_RELEASE_SEM, Release a resource semaphore

Down count (unwind) a resource semaphore's "use count".

## Parameters

*AH*
0x18 (= RTX_RELEASE_SEM)

*BX*
ID of the resource semaphore acquired from **RTX_CREATE_SEM**

## Return Value

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

## Comments

The resource's use count is decremented by one if the calling task presently owns this resource semaphore (otherwise error code -12, "not owner"). The resource is not freed until the use count reaches zero, at which point the caller no longer "owns" this semaphore.

Once freed, the resource will immediately be given to the task (if any) which is waiting at the head of this resource semaphore's wait queue. An immediate task switch occurs if this waiting task is higher **priority** than the calling task that just gave up ownership of this semaphore.

Note: Attempting to use this function on a counting semaphore will fail with error code -12.

## Beck C-Library Usage

**RTX_Release_Sem()**

## Related Topics

**RTX_FREE_RES** Free resource semaphore
**RTX_RESERVE_RES** Reserve resource semaphore
RTOS **Semaphore** Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x19: RTX_RESERVE_RES, Get use of a resource semaphore

Reserves a resource semaphore.

## Parameters

*AH*
0x19 (= RTX_RESERVE_RES)

*BX*
ID of the semaphore acquired from **RTX_CREATE_SEM**

*ES:DI*
Pointer to signed long buffer containing the timeout in milliseconds.
If timeout == 0, the caller waits forever for the resource.
If timeout < 0, the function will return immediately (polling mode). The AX register will be set to error code -27 if the semaphore was not available.

## Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, semaphore is in use

## Comments

This call waits for a defined time to reserve a semaphore and returns with an error code -27 if the semaphore is still in use by another task after the timeout period expires.

The callers wait in FIFO order for the semaphore.

On success, the calling task then owns the resource semaphore. A task which owns the semaphore is free to call here repeated times, reserving the same semaphore more than once. Each such call increments a "use count" internal to the semaphore. This use count must be restored to zero before any other task can be granted ownership of this resource semaphore.

## Beck C-Library Usage

**RTX_Reserve_Sem()**

## Related Topics

**RTX_FREE_RES** Free resource semaphore
**RTX_RELEASE_SEM** Release resource semaphore
**RTX_WAIT_SEM_PRIO** Wait in with specific priority
RTOS **Semaphore** Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x1A: RTX_SIGNAL_SEM, Signal a counting semaphore

Make semaphore access available to one additional task.

## Parameters

*AH*
0x1A (= RTX_SIGNAL_SEM)

*BX*
ID of the semaphore acquired from **RTX_CREATE_SEM**

## Return Value

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

## Comments

Use this function to either surrender access to a counting semaphore, or to indicate that some resource guarded by this semaphore has become available.

Upon this signal, the semaphore will be given to the task (if any) which is waiting at the head of this semaphore's wait queue. This can result in an immediate task switch if this waiting task is higher

priority than the calling task.

This function must not be called for resource semaphores, as this will cause the resource semaphore to malfunction.

## Beck C-Library Usage

RTX_Signal_Sem()

## Related Topics

RTX_WAIT_SEM Wait for counting semaphore access
RTX_GET_SEM Get counting semaphore without waiting
RTOS Semaphore Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x1B: RTX_WAIT_SEM, Wait on a counting semaphore

Wait up to a specified time for access to a counting semaphore.

## Parameters

*AH*
0x1B (= RTX_WAIT_SEM)

*BX*
ID of the semaphore acquired from RTX_CREATE_SEM

*ES:DI*
Pointer to signed long buffer containing the timeout in milliseconds.
If timeout == 0, the caller waits forever for the resource.
If timeout < 0, the function will return immediately (polling mode). The AX register will be set to error code -27 if the semaphore was not available.

## Return Value

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

## Comments

This call waits for a defined time to acquire a counting semaphore and returns with an error code -27 if the semaphore is still not available after the timeout period expires.

The callers wait in FIFO order for the semaphore.

## Beck C-Library Usage

RTX_Wait_Sem()

## Related Topics

RTX_SIGNAL_SEM Signal counting semaphore
RTX_WAIT_SEM_PRIO Wait with specific priority
RTX_GET_SEM Get counting semaphore without waiting
RTOS Semaphore Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x1C: RTX_FIND_SEM, Find semaphore by name

Get Semaphore ID using 4-char name.

## Parameters

*AH*
0x1C (= RTX_FIND_SEM)

*ES:DI*
Pointer to 4 character name tag (no zero terminator needed)

## Return Value

DX =0 success AX: contains the semaphore ID
DX!=0 failure AX: contains error code

## Comments

The semaphore ID obtained here is the handle required for the semaphore API functions.
A semaphore created in some other program can be accessed in this manner.

If more than one semaphore was created with the same tag, you will get back the semaphore ID of one with this tag. But which one is not certain.

## Beck C-Library Usage

RTX_Find_Sem()

## Related Topics

RTOS Semaphore Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x1D: RTX_SLEEP_FINE, Sleep for RTI count

### Parameters

*AH*

0x1D (=RTX_SLEEP_FINE)

*DX:BX*

32 bit signed sleep time specified as a Real-Time Interrupt (RTI) count. This count must be positive.

### Return Value

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

### Comments

Refer to the the **RTX_SLEEP_TIME** API description for information concerning the sleep.

### SC1x Comments

For SC1x targets which have a fixed 1 kHz real-time interrupt rate this API is nearly identical to the **RTX_SLEEP_TIME** API. The only significant difference is that this API accepts a 31 bit sleep count allowing longer sleep periods to be specified.

### SC1x3/SC23 Comments

This API differs from the **RTX_SLEEP_TIME** API when the Real-Time Interrupt (RTI) **rate** is **configured** for greater than 1000 Hz operation.

### Beck C-Library Usage

**RTX_Sleep_Fine()**
**RTX_Sleep_Long()**

### Related Topics

RTOS **Task** Control Services
**RTX_SLEEP_TIME** Sleep for specified milliseconds.
**RTX_SLEEP_REQ** Sleep until wake up call
**RTX_WAKEUP_TASK** Wake up a task
CHIP.INI **RTI** rate selection

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| 1.20 | 1.20 | 1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x1E: RTX_WAIT_SEM_PRIO, Wait on a a semaphore

Wait up to a specified time for access to a semaphore.

## Parameters

*AH*

0x1E (= RTX_WAIT_SEM_PRIO)

*CX*

ID of the semaphore acquired from **RTX_CREATE_SEM**

*ES:BX*

Signed long value containing the timeout in milliseconds
if timeout == 0, the caller waits forever for the resource
If timeout < 0, the function will return immediately (polling mode). The AX register will be set to error code -27 if the semaphore was not available.

*DX*

The priority at which the caller wishes to wait (0 = highest). To wait in FIFO order, have all callers use the same value. (Note that this priority has no connection to task priorities.)

## Return Value

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

## Comments

This function may be used with either counting or resource type semaphores.

This call waits for up to a specified time to acquire the semaphore and returns with an error code -27 if the semaphore is still not available after the timeout period expires.

The callers with higher wait priority (lower number in DX register) cut in line ahead of other tasks waiting for the same semaphore at a lower wait priority (higher number in DX).

## Beck C-Library Usage

**RTX_Wait_Sem()**

## Related Topics

**RTX_SIGNAL_SEM** Signal counting semaphore
**RTX_RESERVE_RES** Wait in FIFO order

RTX_WAIT_SEM Wait in FIFO order
RTX_GET_SEM Get counting semaphore without waiting
RTOS Semaphore Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x20: RTX_DISABLE_TASK_SCHEDULING, Task Lock

Task switching is inhibited until follow up call(s) to RTX_ENABLE_TASK_SCHEDULER is made. Interrupt service routines continue to execute, however Timer procedures will be delayed until task switching is re-enabled.

## Parameters

*AH*
0x20 (= RTX_DISABLE_TASK_SCHEDULING)

## Comments

This function locks the processor to the current task, preventing further task switching until either RTX_ENABLE_TASK_SCHEDULER or a sleep API function is called. Entering any sleep API defeats the task lock. In this case the task lock condition is released until the task resumes execution. During the sleep, the system automatically releases the lock condition to allow other tasks to execute.

During the task lock condition, interrupts still execute provided that the user leaves interrupts enabled. Installed Timer procedures will not be executed. After making this call, the task must follow up with a call to RTX_ENABLE_TASK_SCHEDULER.

Note that this is implemented as a spin lock, such that if for some reason the task calls here N times then N calls to RTX_ENABLE_TASK_SCHEDULER are required to unwind the spin lock and re-enable the task switching.

**Caution:** This call must be followed by a call to RTX_ENABLE_TASK_SCHEDULER as soon as possible to re-enable the task switching. Should the task lock period be excessive, the system watchdog must be triggered by the user until the task switching is re-enabled.

## Beck C-Library Usage

RTX_Disable_Task_Scheduling()

## Related Topics

RTOS Task Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x21: RTX_ENABLE_TASK_SCHEDULING, Release Task Lock

Unwinds the task switching spin lock to re-enable task switching.

## Parameters

*AH*

0x21 (= RTX_ENABLE_TASK_SCHEDULING)

## Comments

This API reverses the affect of the **RTX_DISABLE_TASK_SCHEDULER** API.

When task switching is re-enabled, an immediate task switch may result if there is an active task with higher **priority** than the task making this API call.

## Beck C-Library Usage

**RTX_Enable_Task_Scheduling()**

## Related Topics

API function **RTX_DISABLE_TASK_SCHEDULER**
RTOS **Task** Control Services

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x28: RTX_GET_TIMEDATE, Get system time and date

## Parameters

*AH*

0x28 (= RTX_GET_TIMEDATE)

*BX:SI*

Output parameter: Pointer to `TimeDate_Structure` **type** allocated by user.

## Return Value

DX=0 success AX: 0, Location at [BX:SI] contains system date and time

## Beck C-Library Usage

RTX_Get_TimeDate()

## Related Topics

Set Time/Date
RTOS Time/Date Services
TimeDate_Structure type definition

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## 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.

**Caution:** Values for time/date supplied by the caller are not checked for validity.

## Beck C-Library Usage

RTX_Set_TimeDate()

## Related Topics

Get Time/Date
RTOS Time/Date Services
TimeDate_Structure type definition

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x2A: RTX_GET_TICKS, Get tick count of system clock

Reads out the system millisecond clock tick count.

## Parameters

*AH*

0x2A (= RTX_GET_TICKS)

*BX:SI*

Output Parameter: Pointer to an unsigned long where the tick count will be stored.

## Return Value

Caller's unsigned long at [BX:SI] contains system tick count.

## Comments

The system clock runs at 1000 Hz. So each tick represents 1 millisecond.

## SC1x3/SC23 Comments

Optionally, the system's real-time interrupt can be run at rates higher than 1000 Hz. Nevertheless, this function will return a count representing the number milliseconds that the system has been executing.

## Beck C-Library Usage

**RTX_Get_System_Ticks()** - Uses output parameter pointer
**RTX_GetTickCount()** - Returns the 32 bit tick count

## Related Topics

RTOS **Time/Date** Services
**RTX_RTI_COUNT** Get Real-Time Interrupt count

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list

Index page

## Interrupt 0xAD service 0x2B: RTX_RTI_COUNT, Get real-time interrupt count.

Reads out the system real-time interrupt (RTI) tick count.

### Parameters

*AH*
0x2B (= RTX_RTI_COUNT)

### Return Value

DX:AX contains the unsigned long Real-Time Interrupt count.
CX = RTI ticks per millisecond (= kHz rate of RTI)

### SC1x3/SC23 Comments

The SC1x3/SC23 system clock can optionally be run at rates above 1000 Hz. This API returns the 32 bit count on this interrupt driven clock in the DX:AX register pair. This count rolls over back to zero at 2**32 counts.

The configured interrupt rate in kHz is indicated in the returned CX register. Total elapsed time in milliseconds can be computed by multiplying the RTI count in DX:AX by the rate in CX.

### Beck C-Library Usage

**RTX_Get_RTI_Ticks()**
**RTX_Get_RTI_Rate()**

### Related Topics

Concerning **RTI** rate adjustments
**CHIP.INI** RTI configuration
**RTX_GET_TICKS** Get millisecond count

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

## Interrupt 0xAD service 0x2F: RTX_TIMER_FINE, Install a fine precision timer procedure

Install a fine resolution timer procedure that will be periodically executed by the kernel.

### Parameters

*AH*
0x2F (= RTX_TIMER_FINE)

*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

**This API is only available for SC1x3/SC23 systems.**

The interval here is specified in Real-Time Interrupt (RTI) count, which can differ from milliseconds when the **RTI** rate has been **configured** for higher than 1000 Hz operation.

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()`.

## Beck C-Library Usage

**RTX_Fine_Timer()** - Timer defined in a structure
**RTX_Fine_TimerP()** - Parameters passed directly

## Related Topics

`TimerProc_Structure` **definition**
RTOS **Timer** Procedures
Install **Timer** with millisecond interval

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## 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()`.

## Beck C-Library Usage

**RTX_Install_Timer()** - Timer defined in a structure
**RTX_Install_TimerP()** - Parameters passed directly

## Related Topics

`TimerProc_Structure` **definition**
RTOS **Timer** Procedures
Install Timer with **fine** resolution

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## 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 safe to call this API from within the timer procedure being removed.

It is possible to reinstall a timer procedure after removing it from the system.

## Beck C-Library Usage

**RTX_Remove_Timer()**

## Related Topics

RTOS **Timer** Procedures

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x32: RTX_START_TIMER, Start periodic timer procedure

Starts the 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.

## Comments

The first execution of this timer procedure will occur within one millisecond of this call, after the subsequent @CHIP-RTOS 1000 Hz real-time interrupt.

The user is free to make this call on a timer which has already been started, the affect being that the

phase of the periodic timer callback is shifted to within one millisecond of the call to this API.

## Beck C-Library Usage

RTX_Start_Timer()

## Related Topics

RTOS Timer Procedures
Start Timer with Delay
Stop Timer

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x33: RTX_STOP_TIMER, Stop execution of a timer procedure

Stops 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.

## Comments

The timer procedure can later be restarted.

## Beck C-Library Usage

RTX_Stop_Timer()

## Related Topics

RTOS Timer Procedures
Start Timer
Start Timer with Delay

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x34: RTX_TIMER_DELAY, Set timer delay

Set specified timer's delay period until next callback.

## Parameters

*AH*
0x34 (= RTX_TIMER_DELAY)

*DX*
timerID produced by the **RTX_INSTALL_TIMER** call

*ES:BX*
32 bit delay period count. A zero delay period will stop the timer.

## SC1x3/SC23 Parameters

*CX*
Boolean flag, set non-zero if the delay count in ES:DX is Real-Time Interrupt (RTI) ticks instead of milliseconds.

## Return Value

DX =0 success AX: 0
DX!=0 failure AX: contains **error** code, invalid timerID.

## Comments

The value in ES:BX is not a pointer. This is a 32 bit count with MSH carried in ES register and LSH in BX register.

This function can be used on periodic timers to set the interval to the next call, thereby providing a phase control.

For non-periodic timers this function can be called with varying delay values.

When the delay period in ES:DX is set to '1', this function is identical to the **RTX_START_TIMER** API.

When the delay period in ES:DX is set to '0', this function is identical the **RTX_STOP_TIMER** API.

## SC1x3/SC23 Comments

When the Real-Time Interrupt rate is **configured** for operation above 1000 Hz, the Boolean flag carried in CX becomes relevant.

## Beck C-Library Usage

RTX_Timer_Delay()
RTX_Timer_Delay_RTI()

## Related Topics

RTOS Timer Procedures
Start Timer
Stop Timer

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x35: RTX_SET_RTI_HOOK, Install Real-Time Interrupt hook

This function installs a callback function which executes perioidically within the 1000 Hz (default rate) Real-Time Interrupt (RTI).

## Parameters

*AH*
0x35 (= RTX_SET_RTI_HOOK)

*ES:BX*
Far vector to callback function. Set ES to null to remove the current callback.

*CL*
Number of ticks before first call.
If this value is zero (special case), then the initial delay will be 256 ticks. Set to 1 to execute on next RTI.

## Return Value

-- none --

## Comments

Interrupts are disabled and then restored to their state at the time of the call.

The term 'tick' used here refers to a 1 millisecond period when the system timer RTI is operated at its default rate of 1000 Hz. Note that this RTI rate can be configured to higher rates with the [TIMER] RTI setting in the CHIP.INI, in which case tick time will be under 1 millisecond.

This API was provided for applications which require a precise and variable timing for high frequency front end hardware sampling, for example.

The callback function is free to use any register except for the DS and DI registers. These two registers must be preserved. The value returned in AL from the callback function will be used as the delay count in milliseconds to the next callback. For example, return 1 to execute again on very next RTI. Return zero

for a delay of 256 ticks.

The system supports only a single RTI hook. Installing a second overwrites the first. Unlike with the **RTX_INSTALL_TIMER** timer callback functions, which execute after a task switch to the kernel task, this RTI hook executes in the context of the interrupt. No task switch occurs prior to the callback and execution proceeds on the stack of what ever task was interrupted. Consequently the callback should be conservative with its stack space usage.

This interrupt has been registers with the RTX kernel, so the user is free to use the task triggering RTX API functions from within the callback, such as:

**RTX_WAKEUP_TASK**
**RTX_RESUME_TASK**
**RTX_SIGNAL_SEM**
**RTX_SIGNAL_EVENTS**
**RTX_SEND_MSG**

However any time consuming activity must be avoided and it is not legal to call any RTOS API which could block or result in current task being suspended. As a rough guideline, the dwell within this callback should be held to under 50 us. Either very carefully written C code, or assembly code is recommended here.

As true for any installed callback function, it is essential that an application program remove its RTI hook callback (ES set to zero) before program termination.

## Beck C-Library Usage

**RTX_Install_RTI_Hook()**

## Related Topics

RTOS API **RTX Timers**
**CHIP.INI** RTI configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x40: RTX_CREATE_EVENTGROUP, Create an event group

Creates a new event group of 16 user definable event flags.

## Parameters

*AH*

0x40 (= RTX_CREATE_EVENTGROUP)

*BX*

Initial value of the 16 event flags of the group.

*ES:DI*

Output Parameter: Pointer to 16 bit storage where Event Group ID is output by this API.

*DS:SI*

Pointer to unique four character tag, which need not be a zero terminated string but must consists of four bytes.

## 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 provides 16 Boolean event flags, encoded in a 16 bit word. Users are free to assign any meaning to these Booleans that suites their application.

Event groups can provide control and communicate between programs. One program creates the event group and the other programs must know the event group's unique 4 character tag. The other programs can then obtain the handle to this event group following its creation using the **RTX_FIND_EVENTGROUP** API.

## Beck C-Library Usage

**RTX_Create_EventGroup()**

## Related Topics

RTOS **Event** Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x41: RTX_DELETE_EVENTGROUP, Delete an event group

Deletes specified 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 must not delete an event group which is in use by another task or timer procedure, as this can lead to unpredictable results.

## Beck C-Library Usage

**RTX_Delete_EventGroup()**

## Related Topics

RTOS **Event** Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x42: RTX_SIGNAL_EVENTS, Signal event(s) in a group

Set or clear up to 16 events under a mask in a group.

## Parameters

*AH*
0x42 (= RTX_SIGNAL_EVENTS)

*BX*
Event group ID acquired by **RTX_CREATE_EVENTGROUP** call.

*CX*
16-Bit event group editing mask. The '1' bits here mark event flags to be set or cleared based on the corresponding bit value supplied in DX. Other flags in the event group are unaffected.

*DX*
New event values for the 16 event flags. Only the bits marked '1' in the CX mask are relevant here.

## Return Value

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

## Comments

The Event Manager wakes up any tasks that are waiting on these events and satisfied with the resulting collection of event states (in case of AND waiting condition). This can lead to an immediate task switch if

any of these released tasks are higher **priority** than the signaling task.

## Beck C-Library Usage

**RTX_Signal_Events()**

## Related Topics

RTOS **Event** Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x43: RTX_WAIT_EVENTS, Wait for events in a group

The calling task waits for up to a specified number of milliseconds for event(s) in an event group to assume specified value(s).

## 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 filled in by caller.

## Return Value

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

## Comments

The user must fill in the `RTX_Wait_Event` **structure** before making this call.

The caller can select whether to await all specified events or a single event.

## Beck C-Library Usage

**RTX_Wait_For_Event()** - Uses pointer to data structure
**RTX_Wait_For_EventP()** - Passes direct parameters

## Related Topics

RTX_Wait_Event type definition
RTOS Event Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x44: RTX_GET_EVENTGROUP_STATE, Read the event states

Returns 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*

Output Parameter: 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

## Comments

Note that this API accesses the *current* event group states.

If instead the states recorded at return from the most recent RTX_WAIT_EVENTS call are desired, the alternate RTX_GET_EVENT_FLAGS API can be used.

## Beck C-Library Usage

RTX_Get_EventGroup_State()

## Related Topics

RTOS Event Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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.

### Beck C-Library Usage

RTX_Get_Saved_Events()

### Related Topics

RTOS Event Manager

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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*
0x46 (= RTX_FIND_EVENTGROUP)

*ES:DI*
Pointer to 4 character name tag. This string need not be zero terminated but most be four bytes length.

## Return Value

DX = 0 success AX: contains the event group ID
DX != 0 failure AX: contains **error** code, not found

## Comments

This function allows event groups to be accessed by another program which knows the agreed upon name for the group, thereby providing a communication and control mechanism between programs.

## Beck C-Library Usage

**RTX_Find_EventGroup()**

## Related Topics

RTOS **Event** Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x50: RTX_CREATE_MSG, Create a Message Exchange

Creates a new 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 must fill in portions of the `RTX_Msg` **structure** prior to calling here.

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, thereby allowing other programs access to it by **name**.

For SC1x @CHIP-RTOS versions, the maximum number of Message Exchanges supported by the system is ten. The SC1x3/SC23 @CHIP-RTOS support up to 32 Message Exchanges.

## Beck C-Library Usage

RTX_Create_Msg()

## Related Topics

RTX_Msg type definition
RTOS Message Exchange Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAD service 0x51: RTX_DELETE_MSG, Delete a Message Exchange

Deletes specified 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 must not delete a Message Exchange which is in use by another task or timer procedure, as this may result in unpredictable faults.

## Beck C-Library Usage

RTX_Delete_Msg()

## Related Topics

RTOS Message Exchange Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## 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. This will result in an immediate task switch if the task receiving the message is higher **priority** than the task that is sending the message.

The format of the 12 byte message being sent is defined by the application program. These 12 bytes can contain a pointer to further data, if required. This API copies these 12 bytes into an internal message envelope, so the message at [ES:DI] need not persist beyond the call to this API. (However, any data referenced by pointers contained in the message would, of course, need to be maintained until the message is received by some task.)

The message mailbox priority in CX is applicable for cases where messages are accumulating on the exchange for which there are no immediate waiting message consumer tasks. In this case, a message from the highest priority non-empty mailbox FIFO queue will be given to the next caller of either **RTX_GET_MSG** or **RTX_WAIT_MSG**.

## Beck C-Library Usage

**RTX_Send_Msg()**

## Related Topics

RTOS **Message** Exchange Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAD service 0x53: RTX_GET_MSG, Poll Message Exchange

Get an available message from a specified Message Exchange without waiting.

## Parameters

*AH*

0x53 (= RTX_GET_MSG)

*BX*

Message Exchange ID acquired by **RTX_CREATE_MSG** call.

*ES:DI*

Output Parameter: 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 always returns immediately. If no message is currently available, DX is non-zero and AX = -28.

On success, a message from the highest priority non-empty FIFO mailbox is copied into the 12 byte store at [ES:DI] and then this delivered message is removed from the mailbox.

## Beck C-Library Usage

**RTX_Get_Msg()**

## Related Topics

RTOS **Message** Exchange Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAD service 0x54: RTX_WAIT_MSG, Wait for a message

Wait up to a specified number of milliseconds for a message from a Message Exchange.

### 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.

On success, the highest priority message available on the exchange is copied into your message container at location specified by the `msg` member of the `RTX_Wait_Msg` structure. Then the message is removed from the Message Exchange.

Each caller to this API can specify their priority for access to messages. To wait in FIFO order, all callers have to wait with the same priority.

### Beck C-Library Usage

**RTX_Wait_For_Msg()** - Uses pointer to a data structure
**RTX_Wait_For_MsgP()** - Passes parameters directly

### Related Topics

`RTX_Wait_Msg` **type** definition
RTOS **Message** Exchange Manager

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

This API can be used to access a Message Exchange created with an agreed upon name by another program, thereby providing a communication path between programs.

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.

## Beck C-Library Usage

**RTX_Find_Msg()**

## Related Topics

RTOS **Message** Exchange Manager

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

End of document

---

## RTX_Get_RTI_Rate

Get system's Real-Time Interrupt (RTI) rate in kHz.

```
unsigned int RTX_Get_RTI_Rate (void);
```

## Return Value

Real-Time interrupt rate in kHz

## SC1x Comments

For SC1x systems which always operate at 1 kHz RTI rate, this API replaced by a macro in the library header file defined as a fixed value of '1'.

## SC1x3/SC23 Comments

On SC1x3/SC23 systems the RTI rate can be configured with the CHIP.INI for rates above 1 kHz. This function reads out the configured setting.

## See Also

RTX_Get_RTI_Ticks

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_GetTick_us

Get system time resolved to microseconds.

```
unsigned long RTX_GetTick_us (void);
```

## Return Value

System time in microseconds

## Comments

This function converts the high resolution system time data collected by the **RTX_GetFineTick** API to microseconds and returns this unsigned long 32 bit count value.

Note that this count rolls over to zero after each 2**32 microseconds of operation, which is about 71.582 minutes.

If the **RTX_FineTickInit** API has not been called, then this call will be made here one time. Making this additional API call is time consuming and results in the interrupts being enabled. Therefore calling **RTX_FineTickInit** prior to using this API may be advisable for time critical applications.

Once this initialization has been handled by the **RTX_FineTickInit** API, this `RTX_GetTick_us()` API function is then suitable for use within an Interrupt Service Routine. Interrupts are masked here and then restored to their state on entry. Note that no call into the @Chip-RTOS API is required by this API following the initialization step.

Refer to **RTX_GetFineTick** description for more details.

## SC1x3/SC23 Comments

**CAUTION:**

If the system's real-time interrupt has been **configured** for kHz **rates** which do not divide into 1000 an integral number of times, then this API is inaccurate due to fixed point arithmetic truncation problems and therefore should not be used. The problem arises with the `_us_per_RTI` **constant** used here, computed as:

```
_us_per_RTI = 1000 / _RTI_kHz
```

So for RTI rates like 15 kHz this integer arithmetic truncates and the results from this API are not accurate.

The following RTI settings allow this API to function:

[1, 2, 4, 5, 8, 10, 20, 25, 40, 50]

Any RTI settings from the following set will cause this API to ***not work properly*** :

[3, 6, 12, 15, 16, 24, 30, 32, 48]

## See Also

RTX_FineTickInit  RTX_GetFineTick
RTX_Get_RTI_Ticks  RTX_GetTickCount

## RTOS API

After RTX_FineTickInit has been invoked a single time, this library function makes no entry into the RTOS.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---




# @CHIP-RTOS - SC1x3 RTI Rate Adjustment



## RTI Rate Adjustment on SC1x3/SC23

The SC1x3/SC23 Real-Time Interrupt (RTI) rate can be increased above its default rate of 1 kHz up to a maximum rate of 50 kHz. This configuration is controlled by the CHIP.INI file "[TIMER] **RTI**" entry.

## When to use a higher RTI rate

For applications that require higher resolution scheduling it makes sense to raise the system's clock interrupt rate. The following additional C-library functions can then be used to take advantage of the resulting increased scheduling resolution.

**Task Scheduling**

**RTX_Sleep_Fine** - Sleep a specified number of RTI
**RTX_Sleep_Long** - Sleep a specified long number of RTI

**High Resolution RTX Timers**

**RTX_Fine_Timer** - Creates a high resolution timer
**RTX_Fine_TimerP** - Creates a high resolution timer based on parameters
**RTX_Timer_Delay_RTI** - Specifies delay to next timer callback in RTI ticks

## When *not* to use a higher RTI rate

If your application needs an accurate sense of time, this alone is not a justification for increasing the system's real-time interrupt rate. The system time resolved to microseconds is available from the C-library functions. These functions are available on all @Chip-RTOS platforms and they do not require that the real-time interrupt rate be raised above the default 1 kHz rate.

**High Resolution System Time**

**RTX_GetTick_us** - Get system time with microsecond resolution
**RTX_GetFineTick** - Get high resolution system time

## CPU loading considerations with higher RTI rate

It should not be surprising that the load placed on the CPU increases when the RTI rate is increased. What

loads down the CPU in particular is when **tasks** or RTX **timer** callbacks become due at a high rate. For example if a timer callback is scheduled for each RTI tick, then the CPU quickly becomes swamped by this activity alone as can be seen by the example measurements in the table below.

On the other hand, a high RTI rate with sparsely scheduled tasks does not load the CPU that much.

The following table of measurements may help provide a general idea what rates of activity you can expect to achieve when raising the system's RTI rate. These measurements were made with the Debug@Chip PROBE executing concurrently with a simple test program which installed an RTX timer to be called at specified RTI tick intervals.

| RTI Rate | Idle | 2 ticks | 1 tick |
|---|---|---|---|
| 1 kHz | 5,5% | 7,7% | 9,4% |
| 2 kHz | 5,7% | 9,7% | 13% |
| 5 kHz | 6,1% | 15% | 25% |
| 10 kHz | 6,6% | 24% | 42% |
| 15 kHz | 7,6% | 33% | 59% |
| 20 kHz | 8,0% | 43% | 77% |
| 30 kHz | 9,5% | 61% | dead |
| 40 kHz | 11.0% | 71% | dead |
| 50 kHz | 12,5% | 93% | dead |

The Idle column in the table indicates the load with only the debugger's PROBE executing prior to starting the test program that operates the RTX **timer**. The "2 ticks" column is for the case where the timer interval is set to two RTI ticks. Likewise, for the "1 ticks" column the timer callback is execued on every RTI tick.

Note that you can configure a system which will appear to not start (from a TCP/IP point of view) by setting the RTI rate high enough and starting a program from the **AUTOEXEC.BAT** which configures a timer at too high a callback rate. This was the case for trials in the above table where a "dead" indication appears. So some care will be required when operating at higher RTI rates.

**C-Library RTX Timing API**

---

End of document

---

# @CHIP-RTOS - Web server CGI interface

## CGI API

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.
CGI at the IPC@CHIP® provides the possibility to install at the IPC@CHIP® Web server own programmed CGI functions. These functions are bound with a fixed name and are executed by the Web server task, if a http request with such a fixed name comes in. This mechanism allows dynamic usage of the IPC@CHIP® Web server. For better understanding of CGI and possibilities for using, see description of available program examples at the example link below.

File Upload:
Also there is a possibility to upload files to the @CHIP's disk. This does not use the CGI interface but the Web server. Therefore you only need a defined HTML form which sends the required data to the @CHIP Web server. An example can be found in the CGI Example Zip file.

**Notes:**

1. For test and demonstration purpose two pages are preinstalled at the IPC@CHIP®:
   a)**main.htm:** Static html introduction page.
   b)**chipcfg** **:** Dynamic page with system time/date and configuration data of the requested IPC@CHIP®.
2. For configuring the webserver and CGI, see the provided chip.ini entries.
3. For some useful comments see also under Programming notes

## Topics

Web Server Overview
Web Server FileTypes

CGI API Error Codes
CGI API ExamplesAvailable
CGI API Data Structures

## Interrupt 0xAB - CGI Services

Service is selected by index in AH register.

- 0x01: _CGI_INSTALL, Install a CGI function
- 0x02: _CGI_REMOVE, Remove a CGI function
- 0x03: _CGI_SETMAIN, Set a new main page
- 0x04: _CGI_SETROOTDIR, Set Web server's root directory
- 0x05: _CGI_GETROOTDIR, Get Web server's root directory

- 0x06: _CGI_GETMAIN, Get main page name
- 0x07: _CGI_GETFORMITEM, Split a form item into name and value
- 0x08: _CGI_FINDNEXTITEM, Return the address of the next form item
- 0x09: _CGI_INSTALL_PAS, Install a Turbo Pascal CGI procedure
- 0x0A: _CGI_ERR_FUNC, Install a user CGI function on specified HTTP errors
- 0x0B: _CGI_MIME_INSTALL, Install a custom user MIME table
- 0x0C: _CGI_MIME_REMOVE, Remove the custom user MIME table
- 0x0D: _CGI_CONTENT_ENCODING_INSTALL, Install a user content encoding table
- 0x0E: _CGI_SETMAIN_POST, Set a new main page (POST method)
- 0x0F: _CGI_GETMAIN_POST, Get main page name (POST method)
- 0x10: _CGI_SETROOTDRIVE, Set Web server's root directory
- 0x11: _CGI_GETROOTDRIVE, Get Web server's root directory

## Interrupt 0xAB - SSI Services

Service is selected by index in AH register.

- 0x81: _SSI_SET_FILE_FILTER, Set SSI file filter
- 0x82: _SSI_REGISTER_TAG, Register a SSI tag string
- 0x83: _SSI_SET_TAG_LEN, Set the maximum SSI tag length
- 0x84: _SSI_SEND_BUFFER, Transmit a user buffer
- 0x85: _SSI_DEREGISTER_TAG, Deregister a SSI tag string and handler

---

## Interrupt 0xAB service 0x01: CGI_INSTALL, Install a CGI function

### Parameters

*AH*

0x01 (= CGI_INSTALL)

*DX:SI*

Pointer to a `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 no copy of the information in the provided `CGI_Entry` structure, so the contents of structure at [DX:SI] must be persistent, but not the structure itself. This means that the PathPtr and CgiFuncPtr must be persistent.

The maximum number of available CGI pages is configurable at the chip.ini file, see CGI entries.

### Related Topics

CGI API Error Codes
`cgistat` command line
CGI_INSTALL_PAS API Function, for Pascal CGI Procedures

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

CGI API **Error** Codes
`cgistat` **command** line

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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). The root directory is also configurable at the chip.ini file, see **ROOTDIR** and **DRIVE**.

## Related Topics

**CGI_GETROOTDIR** API Function
CGI API **Error** Codes

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAB service 0x07: CGI_GETFORMITEM, Split a form item 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 to pick up additional form items, 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.

## SC1x Comments

Method "GET":
For preventing internal buffer overruns, the user should provide 64 bytes of buffer addressed by `FormItem.NamePtr` and 256 bytes pointed by `FormItem.ValuePtr`. The max. size for a full pathname (pagename+complete argumentbuffer)
e.g. `"example?Name1=Value1&Name2=Value2"`
is 256 bytes.
Because of this behaviour, the storage addressed by `FormItem.ValuePtr` must have the max. size of a full pathname (256 bytes).
See example submit.c.

Method "POST":
The size of the argument working buffer is configurable with the chip.ini setting **POST_SIZE**. Default value is 768 bytes.
For preventing internal buffer overruns, the user should provide 64 bytes of buffer addressed by `FormItem.NamePtr` and "POST_SIZE" bytes pointed by `FormItem.ValuePtr`. The storage addressed by `FormItem.ValuePtr` must have the max. possible size "POST_SIZE".
See **example** submitp.c.

## SC1x3/SC23 Comments

Method "GET" and "POST":
The max. length of the whole http header is configurable with the chip.ini setting **HEADER_SIZE**. For preventing internal buffer overruns, the user should provide 64 bytes of buffer addressed by `FormItem.NamePtr` and "HEADER_SIZE" bytes pointed by `FormItem.ValuePtr`. The storage addressed by `FormItem.ValuePtr` must have the max. size "HEADER_SIZE".
See **example** submitp.c.

## Related Topics

**CGI_FINDNEXTITEM** API Function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

# Interrupt 0xAB service 0x08: CGI_FINDNEXTITEM, Return the address of the next form item

Most forms 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | n/a |

Top of list
Index page

## Interrupt 0xAB service 0x0A: CGI_ERR_FUNC, Install a user CGI function on specified HTTP errors

This functions allows the installation of user CGI functions, if a HTTP error occurs, e.g. HTTP 404 Not found. It is possible to install different CGI handlers.

### Parameters

*AH*

0x0A (= CGI_ERR_FUNC )

*BX*

Http errorstate

07: 401 Unauthorized
08: 404 Not found
10: 500 Server error (File system error)
22: 503 Service not available (Out of memory)

*ES:DI*

CGI Function pointer

### Return Value

DX=0 AX=0, success
DX=-1 AX=-4, no space left at internal CGI error handler table

### Comments

The CGI function must be implemented in the same way than any "normal CGI function", installed with **CGI_INSTALL**.
For deleting an entry, install a null pointer (es=di=0).

Please note: Do not install for HTTP state 401 (authentication reqired) a CGI function, which does a redirect to another file. This blocks the input of the username and password by a browser.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V1.12 | V1.00 |

**Top of list**
**Index page**

## Interrupt 0xAB service 0x0B: CGI_MIME_INSTALL, Install a custom user MIME table

This functions allows the installation of user specific MIME types.

### Parameters

*AH*

0x0B (= CGI_MIME_INSTALL )

*BX*

Number of entries in MIME table

*ES:DI*
Pointer to **MIME table**

## Return Value

DX=0 AX=0, success
DX!=0, error

## Comments

The MIME table must be persistent. The RTOS does not make a copy.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAB service 0x0C: CGI_MIME_REMOVE, Remove the custom user MIME table

This functions allows the deinstallation of user specific MIME types.

## Parameters

*AH*
0x0C (= CGI_MIME_REMOVE )

## Return Value

DX=0 AX=0, success
DX!=0, error

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAB service 0x0D: CGI_CONTENT_ENCODING_INSTALL, Install a user content encoding table

This functions allows the installation of user specific content-encoding types.

## Parameters

*AH*

0x0D (= CGI_CONTENT_ENCODING_INSTALL )

*BX*

Number of entries in content-encoding table (must be equal to the user custom mime table size).

*ES:DI*

Pointer to **ContentEncoding table**

## Return Value

DX=0 AX=0, success
DX!=0, error

## Comments

The content encoding table references to the custom MIME table. So the content encoding table assigns every mime type defined in the MIME table a correspondent content encoding type.
To remove an installed content-encoding table use in register BX the value 0.
The content encoding table must be persistent. The RTOS does not make a copy.

## Related Topics

**CGI_MIME_INSTALL** API Function, to install a custom user MIME table

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.16 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAB service 0x0E: CGI_SETMAIN_POST, Set a new main page (POST method)

## Parameters

*AH*

0x0E (= CGI_SETMAIN_POST)

*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_POST** API Function
CGI API **Error** Codes

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.20 | V1.20 |

**Top of list**
**Index page**

---

## Interrupt 0xAB service 0x0F: CGI_GETMAIN_POST, Get main page name (POST method)

## Parameters

*AH*
0x0F (= CGI_GETMAIN_POST)

## 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_POST** API Function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.20 | V1.20 |

**Top of list**
**Index page**

---

## Interrupt 0xAB service 0x10: CGI_SETROOTDRIVE, Set Web server's root directory

## Parameters

*AH*
0x10 (= CGI_SETROOTDRIVE)

*DX*
New root drive (00h = A:, 01h = B:, etc)

## Return Value

DX =0 success AX: 0
DX!=0 failure AX: error code CGI_INVALID_DIR

## Comments

The root drive is also configurable at the chip.ini file, see **ROOTDIR** and **DRIVE**.

## Related Topics

**CGI_GETROOTDRIVE** API Function
CGI API **Error** Codes

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.26 | V1.26 | V1.26 | V1.21 | V1.21 |

**Top of list**
**Index page**

---

## Interrupt 0xAB service 0x11: CGI_GETROOTDRIVE, Get Web server's root directory

## Parameters

*AH*
0x11 (= CGI_GETROOTDRIVE)

## Return Value

AX=0 , DX contains the root drive (00h = A:, 01h = B:, etc)

## Comments

The value in DX contains the webservers root drive (00h = A:, 01h = B:, etc)

## Related Topics

**CGI_SETROOTDRIVE** API Function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.26 | V1.26 | V1.26 | V1.21 | V1.21 |

**Top of list**
**Index page**

## Interrupt 0xAB service 0x81: SSI_SET_FILE_FILTER, Set SSI file filter

Set Server Side Include (SSI) file filter

### Parameters

*AH*

0x81 (= SSI_SET_FILE_FILTER)

*ES:DI*

Pointer to null-terminated string depending on the machted type

*BX*

Match type, possible values are:

```
SSI_MATCH_FILE_NAME(0x01) = File name, Example: "table.htm"
SSI_MATCH_FILE_EXT (0x02) = File extension, Example: "htm"
SSI_MATCH_FILE_PATH(0x03) = Full path to file, Example: "/web/ssi/table.htm"
SSI_MATCH_DIR_NAME (0x04) = Directory name, Example: "ssi"
SSI_MATCH_DIR_PATH (0x05) = Directory path, Example: "/web/ssi"
```

*CX*

0 = register for HTTP server
1 = register for HTTPS (SSL) server

### Return Value

DX =0 success AX: 0
DX=-1 failure AX: contains error code CGI_INVALID_METHOD

### Comments

SSI commands are special HTML comment tags in web pages, and parsed by the web server while the pages are being served. They allow dynamically generated content to be inserted into a static page, without having to dynamically generate the whole page.
There's no official standard for SSI command. Thus we provide an API to register a SSI tag string and a handler function. When a registered SSI tag is found in a page being served, the handler function will be called to process it.

### Related Topics

SSI_REGISTER_TAG API Function, to register a SSI tag

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

## Interrupt 0xAB service 0x82: SSI_REGISTER_TAG, Register a SSI tag string

Register Server Side Include (SSI) tag string

## Parameters

*AH*

0x82 (= SSI_REGISTER_TAG)

*ES:DI*

Tag string, Pointer to null-terminated string

*BX:SI*

user callback function for this tag.

**Callback function prototype:**

```
typedef
  int huge (far *SSI_Handler) (unsigned long conhandle, const char far *
tagPtr);
```

**Possible callback function return values:**

`SSI_OUTPUT_ENABLE(0x00)` = Enable web server to output static content
`SSI_OUTPUT_DISABLE(0x01)` = Disable web server to output static content
`SSI_ABORT_PAGE(0x02)` = Stop processing the current file
other = The web server will send an error message and stop serving the HTTP request

*CX*

0 = register for HTTP server
1 = register for HTTPS (SSL) server

## Return Value

DX =0 success AX: 0
DX=-1 failure AX: contains error code CGI_INVALID_METHOD or CGI_NO_FREE_ENTRY

## Related Topics

**SSI_SET_FILE_FILTER** API Function, to set a SSI file filter
**SSI_SEND_BUFFER** API Function, to send a buffer

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAB service 0x83: SSI_SET_TAG_LEN, Set the maximum SSI tag length

Set maximum length of Server Side Include (SSI) tag string

## Parameters

*AH*

0x83 (= SSI_SET_TAG_LEN)

*BX*
length in bytes (default is 90)

## Return Value

DX =0 success AX: 0
DX=-1 failure AX: contains error code CGI_INVALID_METHOD

## Comments

Smaller values will optimize the performance of the SSI parser.

## Related Topics

**SSI_REGISTER_TAG** API Function, to register a SSI tag
**SSI_SET_FILE_FILTER** API Function, to set a SSI file filter

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAB service 0x84: SSI_SEND_BUFFER, Transmit a user buffer

Transmit a user buffer to the HTTP client through the HTTP connection

## Parameters

*AH*
0x84 (= SSI_SEND_BUFFER)

*ES:DI*
pointer to user buffer

*BX:SI*
connection handle (32 bit value)

*CX*
buffer length in bytes, max. 32767

*DX*
flag, set to 1

## Return Value

DX =0 success AX: 0
DX=-1 failure AX: contains error code

## Comments

Use this function in the SSI_Handler callback function to replace SSI tags with dynamically generated data.

## Related Topics

SSI_REGISTER_TAG API Function, to register a SSI tag
SSI_SET_FILE_FILTER API Function, to set a SSI file filter

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAB service 0x85: SSI_DEREGISTER_TAG, Deregister a SSI tag string and handler

Deregister Server Side Include (SSI) tag string and handler

## Parameters

*AH*
0x85 (= SSI_DEREGISTER_TAG)

*BX:SI*
user callback function for this tag.

*CX*
0 = deregister for HTTP server
1 = deregister for HTTPS (SSL) server

## Return Value

DX =0 success AX: 0
DX=-1 failure AX: contains error code CGI_INVALID_METHOD

## Related Topics

SSI_SET_FILE_FILTER API Function, to set a SSI file filter
SSI_REGISTER_TAG API Function, to register a tag

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.21 | V1.21 |

Top of list
Index page

## BIOS_Server_On_Off

Suspend/Resume system servers.

```
int BIOS_Server_On_Off ( SERVER_SPEC which_server,
                         unsigned int activate );
```

## Parameters

*which_server*

Enumerator:
SS_FTP_SERVER = 0, SS_TELNET_SERVER = 1, SS_WEB_SERVER = 2,
SS_SSL_WEB_SERVER = 3, SS_SSH_SERVER = 4

*activate*

Boolean: 0: Resume, 1: Suspend.

## Return Value

0: Success
1: Server was already ub the postulated state
-1: Invalid parameters

## Comments

This function may be used to enable a server that was disabled at startup due to a CHIP.INI entry.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

**FTP_ENABLE** definition in CHIP.INI file
**TELNET_ENABLE** definition in CHIP.INI file
**WEB_ENABLE** definition in CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS - Hardware API

IPC@CHIP® Documentation **Index**

## Hardware API

Hardware API Here are the interface definitions for access to the IPC@CHIP®'s hardware.

## Topics

Hardware API **Layer**Model

**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.

For some useful comments see also under **Programming notes**

## PFE Interrupt 0xA2 Services

- **0x80:_PFE: Enable Data Bus**
- **0x81:_PFE: Enable Non-Multiplexed Address Bus**
- **0x82:_PFE: Enable Programmable I/O Pins**
- **0x83:_PFE: Enable Programmable Chip Selects**
- **0x84:_PFE: Enable External Interrupt Requests**
- **0x85:_PFE: Enable External Timer Inputs/Outputs**
- **0x86:_PFE: Set Edge/Level Interrupt Mode**
- **0x87:_PFE: Enable PWD Mode**
- **0x88:_PFE: Enable External DMA**
- **0x89:_PFE: Enable INT0 / INTA cascade mode**
- **0x8A:_PFE: Set wait states for PCS0-3**
- **0x8B:_PFE: Set wait states for PCS5-6, respectively PCS4-7**
- **0x8C:_PFE: Set wait states for UCS**
- **0x8D:_PFE: Enable Bus Signals**
- **0x90:_PFE: Get Hardware API Function Pointers**

## HAL Interrupt 0xA1 Services

- **0x10:_HAL: Set int0 Vector**
- **0x80:_HAL: Read Data Bus**

- 0x81:_HAL: Write Data Bus
- 0x82:_HAL: Read Programmable I/O Pins
- 0x83:_HAL: Write Programmable I/O Pins
- 0x84:_HAL: Install Interrupt Service Routine
- 0x85:_HAL: Initialize Timer Settings
- 0x86:_HAL: Start Timer
- 0x87:_HAL: Stop Timer
- 0x88:_HAL: Read Timer Count
- 0x89:_HAL: Write Timer Count
- 0x8A:_HAL: Get Frequencies
- 0x8B:_HAL: Set Timer Duty Cycle Waveform
- 0x8C:_HAL: Read Specific I/O Pin
- 0x8D:_HAL: Write to Specific I/O Pin
- 0x8E:_HAL: Give EOI
- 0x8F:_HAL: Initialize Timer Settings Ext
- 0x90:_HAL: Get/Set Watchdog Mode
- 0x91:_HAL: Refresh Watchdog
- 0x92:_HAL: Mask/Unmask Int
- 0x93:_HAL: Write Programmable I/O Pins (Extended)
- 0xA0:_HAL: Block Read Data Bus
- 0xA1:_HAL: Block Write Data Bus
- 0xB0:_HAL: Start DMA Mode
- 0xB1:_HAL: Stop DMA Transfer
- 0xB2:_HAL: Get DMA Info
- 0xC0:_HAL: Initialize/Restore Non-Volatile Data
- 0xC1:_HAL: Save Non-Volatile Data
- 0xC2:_HAL: Get Reboot Reason
- 0xC3:_HAL: Install PowerfailDone callback


---

## Interrupt 0xA2 service 0x80: PFE: Enable Data Bus

**SC1x:**
Initialize data bus I/O mask and ALE usage.

**SC1x3:**
Initialize data bus I/O mask

## Parameters

*AH*
Must be 0x80.

## SC1x Parameters

*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)

## SC1x3 Parameters

*AL*

Don't care

*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 15 = 0: Data bus bit 15 is input, 1: is output

## Return Value

none

## Comments

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'.

## SC1x Comments

Used pins:
ALE, AD[0..7], RD#, WR#

Excluded pins:
If ALE is used, then PCS0# is not available.

## SC1x3 Comments

Used pins:
A[0..22], D[0..15], RD#, WR#

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

---

## Interrupt 0xA2 service 0x81: PFE: Enable Non-Multiplexed Address Bus

**SC1x:**

The IPC@CHIP®:s SC12, SC11 and SC13 have three non-multiplexed address bit outputs, A0 through A2. The enabling of these pins is done here.

**SC1x3:**

The IPC@CHIP®s SC123/SC143 have one non-multiplexed address bit output A23, which is also a PIO

pin. The enabling of A23 as an address line is done here.

## Parameters

*AH*

Must be 0x81.

## SC1x Parameters

*DX*

Mask
- Bit 0 = 1 Enable A0
- Bit 1 = 1 Enable A1
- Bit 2 = 1 Enable A2
- Bit 3..15 not used

## SC1x3 Parameters

*DX*

Mask
- Bit 0..14 not used
- Bit 15 = 1 Enable A23

## Return Value

none

## SC1x 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.

## SC1x3 Comments

Used pins:
A23

Excluded pins:
If A23 is enabled then PIO26 is not available

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

Enable used programmable I/O pins. Define which pins are inputs and which are outputs.

## Parameters

*AH*

Must be 0x82.

## SC1x Parameters

*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 mask

Bit 0 = 1 Enable PIO0
Bit 1 = 1 Enable PIO1
:
Bit 13 = 1 Enable PIO13
Bit 14..15 not used (for future extensions)

## SC1x3 Parameters

*AL*

Mode

**PIOs 0-15:**
0 = Only read PIO state
1,2 = Input with pullup
4 = Output init value = High
5 = Output init value = Low
**PIOs 16-24, 26-31:**
10 = Only read PIO state
11,12 = Input with pullup
14 = Output init value = High
15 = Output init value = Low

*DX*

PIO pin mask

Bit 0 = 1 Enable PIO0 / PIO16 (see mode)
Bit 1 = 1 Enable PIO1 / PIO17 (see mode)
:
Bit 15 = 1 Enable PIO15 / PIO31 (see mode)

## SC23 Parameters

*AL*

Mode

**PIOs 0-2, 9-13:**
0 = Only read PIO state
1,2,3 = Input with pullup/down (PIO1/PIO2=pulldown, all other=pullup)
4 = Output init value = High
5 = Output init value = Low
**PIOs 18-23, 27-28, 31:**

10 = Only read PIO state
11,12,13 = Input with pullup
14 = Output init value = High
15 = Output init value = Low

*DX*

PIO pin

Bit 0 = 1 Enable PIO0 (see mode)
Bit 1 = 1 Enable PIO1 (see mode)
Bit 2 = 1 Enable PIO2 / PIO18 (see mode)
Bit 3 = 1 Enable PIO19 (see mode)
Bit 4 = 1 Enable PIO20 (see mode)
Bit 5 = 1 Enable PIO21 (see mode)
Bit 6 = 1 Enable PIO22 (see mode)
Bit 7 = 1 Enable PIO23 (see mode)
Bit 9 = 1 Enable PIO9 (see mode)
Bit 10 = 1 Enable PIO10 (see mode)
Bit 11 = 1 Enable PIO11 / PIO27 (see mode)
Bit 12 = 1 Enable PIO12 / PIO28 (see mode)
Bit 13 = 1 Enable PIO13 (see mode)
Bit 15 = 1 Enable PIO31 (see mode)

## SC1x 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 = 1: PIO13 is output
Bit 14..15 not used (for future extensions)

AX = DX = CX = 0, Error: Wrong arguments

## SC1x3 Return Value

AX = wPIO

Bit 0 = 1: PIO0 / PIO16 is PIO
Bit 1 = 1: PIO1 / PIO17 is PIO
:
Bit 15 = 1: PIO15 / PIO31 is PIO

DX = wINPUTS (all pins, including non-PIO pins)

Bit 0 = 1: PIO0 / PIO16 is input
Bit 1 = 1: PIO1 / PIO17 is input
:
Bit 15 = 1: PIO15 / PIO31 is input

CX = wOUTPUTS (all pins, including non-PIO pins)

Bit 0 = 1: PIO0 / PIO16 is output
Bit 1 = 1: PIO1 / PIO17 is output
:
Bit 15 = 1: PIO15 / PIO31 is output

AX = DX = CX = 0, Error: Wrong arguments

## SC23 Return Value

AX = wPIO
Bit 0 = 1: PIO0 is PIO
Bit 1 = 1: PIO1 is PIO
Bit 2 = 1: PIO2 / PIO18 is PIO
Bit 3 = 1: PIO19 is PIO
Bit 4 = 1: PIO20 is PIO
Bit 5 = 1: PIO21 is PIO
Bit 6 = 1: PIO22 is PIO
Bit 7 = 1: PIO23 is PIO
Bit 9 = 1: PIO9 is PIO
Bit 10 = 1: PIO10 is PIO
Bit 11 = 1: PIO11 / PIO27 is PIO
Bit 12 = 1: PIO12 / PIO28 is PIO
Bit 13 = 1: PIO13 is PIO
Bit 15 = 1: PIO31 is PIO

DX = wINPUTS (all pins, including non-PIO pins)
Bit 0 = 1: PIO0 is input
Bit 1 = 1: PIO1 is input
Bit 2 = 1: PIO2 / PIO18 is input
Bit 3 = 1: PIO19 is input
Bit 4 = 1: PIO20 is input
Bit 5 = 1: PIO21 is input
Bit 6 = 1: PIO22 is input
Bit 7 = 1: PIO23 is input
Bit 9 = 1: PIO9 is input
Bit 10 = 1: PIO10 is input
Bit 11 = 1: PIO11 / PIO27 is input
Bit 12 = 1: PIO12 / PIO28 is input
Bit 13 = 1: PIO13 is input

CX = wOUTPUTS (all pins, including non-PIO pins)
Bit 0 = 1: PIO0 is output
Bit 1 = 1: PIO1 is output
Bit 2 = 1: PIO2 / PIO18 is output
Bit 3 = 1: PIO19 is output
Bit 4 = 1: PIO20 is output
Bit 5 = 1: PIO21 is output
Bit 6 = 1: PIO22 is output
Bit 7 = 1: PIO23 is output
Bit 9 = 1: PIO9 is output
Bit 10 = 1: PIO10 is output
Bit 11 = 1: PIO11 / PIO27 is output
Bit 12 = 1: PIO12 / PIO28 is output
Bit 13 = 1: PIO13 is output

AX = DX = CX = 0, Error: Wrong arguments

## Comments

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).

used pins:
PIO[0..x]
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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA2 service 0x83: PFE: Enable Programmable Chip Selects

**SC1x:**
Enable chip selects PCS[0..3]#, PCS[5..6]#.

**SC1x3:**
Enable chip selects PCS[4..7]#.

## Parameters

*AH*
Must be 0x83.

## SC1x Parameters

*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

## SC1x3 Parameters

*DX*
Mask
Bit 0 = don't care (no PIO), PCS0# is active when I/O address between 000h..1FFh

Bit 1 = don't care (no PIO), PCS1# is active when I/O address between 200h..3FFh
Bit 2 = don't care (no PIO), PCS2# is active when I/O address between 400h..5FFh
Bit 3 = don't care (no PIO), PCS3# is active when I/O address between 600h..7FFh
Bit 4 = 1 Enable PCS4#, active when I/O address between 800h..9FFh
Bit 5 = 1 Enable PCS5#, active when I/O address between A00h..BFFh
Bit 6 = 1 Enable PCS6#, active when I/O address between C00h..DFFh
Bit 7 = 1 Enable PCS7#, active when I/O address between E00h..FFFh
Bit 8..15 = don't care

## Return Value

none

## SC1x Comments

Used pins:
PCS[0..3]#, PCS[5..6]#

Excluded pins:
If PCS0#: ALE (multiplexed address / data bus)
If PCS1#: A0, PIO4, TMRIN0
If PCS2#: PIO6, INT2, INTA#, PWD, hw flow control serial port 1, cascaded interrupt controller
If PCS3#: PIO5, INT4, hw flow control serial port 1
If PCS5#: A[1..2], PIO3, TMROUT1, TMRIN1
If PCS6#: A[1..2], PIO2

## SC1x3 Comments

Used pins:
PCS[0..7]#

Excluded pins:
If PCS4#: PIO4
If PCS5#: PIO3
If PCS6#: PIO2
If PCS7#: PIO5

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

---

## Interrupt 0xA2 service 0x84: PFE: Enable External Interrupt Requests

**SC1x:**
Enable external interrupt requests INT[0], INT[2..6].

**SC1x3/SC23:**
Enable external interrupt requests INT[1], INT[3], INT[5], PFI.

## Parameters

*AH*

Must be 0x84.

## SC1x Parameters

*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

## SC1x3/SC23 Parameters

*DX*

Mask

Bit 0 = don't care
Bit 1 = 1 Enable INT1
Bit 2 = don't care
Bit 3 = 1 Enable INT3
Bit 4 = don't care
Bit 5 = 1 Enable INT5
Bit 6..14 = don't care
Bit15 = 1 Enable PFI (powerfail interrupt)

## Return Value

none

## SC1x Comments

Used pins:

INT0, INT[2..6]

Excluded pins:

If INT0: PIO13, TMROUT0, cascaded interrupt controller
If INT2: PIO6, PCS2#, INTA#, PWD, hw flow control serial port 1
If INT3: PIO12, serial port 1
If INT4: PIO5, PCS3#, hw flow control serial port 1
If INT5: PIO1, DRQ0, default I²C-Bus pins
If INT6: PIO0, DRQ1, default I²C-Bus pins

## SC1x3 Comments

Used pins:

INT1, INT3, INT5, PFI

Excluded pins:

If INT5: PIO30
If PFI: PIO9

## SC23 Comments

Used pins:

INT1, INT3, INT5, PFI

Excluded pins:
If INT5: PIO0 / TMRIN1
If INT3: PIO1 / TMROUT1
If INT1: PIO2
If PFI: PIO9

## Related Topics

Set **Edge/Level** Interrupt Mode

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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).

## 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

If on a given timer the external input is selected, then that timer's external output is not available and vice-versa.

Used pins:
TMRIN[0..1], TMROUT[0..1]

## SC1x Comments

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

## SC1x3 Comments

Excluded pins:
If TMRIN0: PIO11
If TMRIN1: PIO0
If TMROUT0: PIO10
If TMROUT1: PIO1

## SC23 Comments

Excluded pins:
If TMRIN0: PIO11
If TMRIN1: PIO0 / INT5
If TMROUT0: PIO10
If TMROUT1: PIO1 / INT3

## Related Topics

HAL **Initialize** Timer Settings

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA2 service 0x86: PFE: Set Edge/Level Interrupt Mode

**SC1x:**
Set edge/level interrupt mode for INT0, INT2, INT3, INT4.

**SC1x3/SC23:**
Set edge/level interrupt mode for INT1, INT3, INT5.

## Parameters

*AH*
Must be 0x86.

*AL*
1 = level-sensitive interrupt
0 = edge-triggered interrupt

## SC1x Parameters

*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

## SC1x3/SC23 Parameters

*DX*

Mask, bits set to designate interrupts affected:
Bit 0 = don't care
Bit 1 = INT1
Bit 2 = don't care
Bit 3 = INT3
Bit 4 = don't care
Bit 5 = INT5#
Bit 6..15 = don't care

## Return Value

none

## SC1x 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.

Level-sensitive mode for INT5 / INT6 is not supported. The INT5 / INT6 interrupts operate only in edge-triggered mode.

In level-sensitive mode all interrupt lines are active high.
In edge-triggered mode all interrupt lines act on the low-to-high transition.

## SC1x3/SC23 Comments

Default for INT3 interrupt is level-triggered mode. Default for INT1 and INT5# interrupts is edge-triggered mode. In each case (edge or level) the interrupt pins must remain active until they are acknowledged.

Edge-sensitive mode for INT3 is not allowed if the USB controller is used, because the INT3 interrupt is shared with the USB interrupt.

In level-sensitive mode all interrupt lines except INT5# are active high. INT5# is active low.
In edge-triggered mode all interrupt lines except INT5# act on the low-to-high transition. INT5# acts on the high-to-low transition.

## Related Topics

**Enable External** Interrupt Requests

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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 determine 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | n/a |

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.

## SC1x Parameters

*AL*

DRQ channel:
0 = DRQ0
1 = DRQ1

## SC1x3 Parameters

*AL*

DRQ channel:
0 = DRQ0
1 = DRQ1
2 = DRQ2
3 = DRQ3

## Return Value

AX = 0 no error
AX = -1 invalid DRQ channel
AX = -2 DMA channel is used for serial interface

## Comments

The DMA channel is maybe in use by the serial ports. You must check the current **DMA configuration**.

## SC1x Comments

Used pins:
DRQ[0..1]

Excluded pins:
If DRQ0: PIO1, INT5, default I²C-Bus pins
If DRQ1: PIO0, INT6, default I²C-Bus pins

## SC1x3 Comments

Used pins:
DRQ[0..3]

Excluded pins:
If DRQ0: PIO12
If DRQ1: PIO29
If DRQ2: PIO7
If DRQ3: PIO8

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | n/a |

**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 function **0x84** "Install Interrupt Service Routine" can not be used, because
the cascaded interrupt controller supply the interrupt type over the bus.
Install a normal interrupt service routine (ISR) using the `setvect` function.
At the end of your ISR, issue an EOI to both 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, hw flow control serial port 1

## Related Topics

**HAL 0x8E** Give EOI

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | n/a | n/a |

**Top of list**
**Index page**

---

## Interrupt 0xA2 service 0x8A: PFE: Set wait states for PCS0-3

Set wait states for programmable chip selects PCS0#-PCS3#

## Parameters

*AH*

Must be 0x8A.

## SC11/SC13 Parameters

*AL*

Bit 0-4

00000b = 0 wait states
00001b = 1 wait states
00010b = 2 wait states
00011b = 3 wait states
01000b = 5 wait states
01001b = 7 wait states
01010b = 9 wait states
01011b = 15 wait states
10000b = 24 wait states
10001b = 41 wait states
10010b = 74 wait states
10011b = 123 wait states
11000b = 157 wait states
11001b = 175 wait states
11010b = 201 wait states
11011b = 255 wait states

Bit 5-7

don't care

## SC12/SC1x3 Parameters

*AL*

Bit 0-3

0000b = 0 wait states
0001b = 1 wait states
0010b = 2 wait states
0011b = 3 wait states
1000b = 5 wait states
1001b = 7 wait states
1010b = 9 wait states
1011b = 15 wait states

Bit 4-7

don't care

## Return Value

none

## SC12/SC1x3 Comments

Default for PCS0#-PCS3# are 15 wait states.

## SC11/SC13 Comments

Default for PCS0#-PCS3# are 41 wait states.

## Related Topics

**Enable programmable chip selects**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | V1.00 | V0.90 | n/a |

## Interrupt 0xA2 service 0x8B: PFE: Set wait states for PCS5-6, respectively PCS4-7

**SC11/SC13:**
Set wait states for programmable chip selects PCS5# - PCS6#

**SC1x3:**
Set wait states for programmable chip selects PCS4# - PCS7#

## Parameters

*AH*
Must be 0x8B.

## SC1x3 Parameters

*AL*
Bit 0-4
00000b = 0 wait states
00001b = 1 wait states
00010b = 2 wait states
00011b = 3 wait states
01000b = 4 wait states
01001b = 5 wait states
01010b = 6 wait states
01011b = 7 wait states
10000b = 8 wait states
10001b = 9 wait states
10010b = 10 wait states
10011b = 11 wait states
11000b = 12 wait states
11001b = 13 wait states
11010b = 14 wait states
11011b = 15 wait states
Bit 5-7
don't care

## SC11/SC13 Parameters

*AL*
Bit 0-4
00000b = 0 wait states
00001b = 1 wait states
00010b = 2 wait states
00011b = 3 wait states
01000b = 5 wait states
01001b = 7 wait states
01010b = 9 wait states
01011b = 15 wait states
10000b = 24 wait states
10001b = 41 wait states
10010b = 74 wait states
10011b = 123 wait states
11000b = 157 wait states
11001b = 175 wait states
11010b = 201 wait states
11011b = 255 wait states

Bit 5-7
don't care

## Return Value

none

## SC11/SC13 Comments

Default for PCS5# - PCS6# are 9 wait states.

## SC1x3 Comments

Default for PCS4# - PCS7# are 15 wait states.

## Related Topics

**Enable programmable chip selects**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | V1.00 | V1.00 | V0.90 | n/a |

**Top of list**
**Index page**

---

## Interrupt 0xA2 service 0x8C: PFE: Set wait states for UCS

Set wait states for external memory chip select UCS#

## Parameters

*AH*
Must be 0x8C.

*AL*
Bit 0-4
00000b = 0 wait states
00001b = 1 wait states
00010b = 2 wait states
00011b = 3 wait states
01000b = 4 wait states
01001b = 5 wait states
01010b = 6 wait states
01011b = 7 wait states
10000b = 8 wait states
10001b = 9 wait states
10010b = 10 wait states
10011b = 11 wait states
11000b = 12 wait states
11001b = 13 wait states
11010b = 14 wait states
11011b = 15 wait states

Bit 5-7
don't care

## Return Value

none

## Comments

Default for UCS# are 9 wait states.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | n/a |

Top of list
Index page

---

## Interrupt 0xA2 service 0x8D: PFE: Enable Bus Signals

Activate the bus signals ARDY, HOLD, WRL/WRH and A0/BHE.

## Parameters

*AH*
Must be 0x8D.

*BX*
Signal Mask
Bit0=1: Enable Asynchronous Ready (ARDY)
Bit1=1: Enable HOLD
Bit2=1: Enable WRL/WRH
Bit3=1: Enable A0/BHE

## Return Value

none

## Comments

Used pins:
ARDY, HOLD

Excluded pins:
If ARDY: PIO6
If HOLD: PIO17
If WRL/WRH: A0/BHE
If A0/BHE: WRL/WRH

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | n/a |

Top of list
Index page

---

## Interrupt 0xA2 service 0x90: PFE: Get Hardware API Function Pointers

Get the Function Pointers to Hardware API Functions, so that the functions can be called directly (without Software Interrupt). This is much faster.

### Parameters

*AH*

Must be 0x90.

*ES:DI*

Pointer to a `HwApiFunctionStruct` data structure which will be filled with vectors to @CHIP-RTOS Hardware API Functions.

### Return Value

AX= -1 if `size` parameter was too large. The `size` field of the structure will be set in this case to the number of supported vectors.

AX= 0 on success

### Comments

Note that the `size` member of the `HwApiFunctionStruct` serves as both an input and output parameter. The caller must set this value to the number of vectors to be filled into the data structure by this API call. In the event that this value exceeds the number of vectors available, the AX value returned will be -1 and this API will set the `size` to 2, which is the number of vectors available by this API's current implementation.

In any case, the resulting count in `size` indicates the number of vectors output to the caller's `HwApiFunctionStruct` data structure at [ES:DI] by this API call.

```
typedef struct
{
   int size; // number of function entries
   unsigned  (huge *readPios)(void);
   void      (huge *writePios)(unsigned);
} HwApiFunctionStruct;

// Example for usage:

int main()
{
  HwApiFunctionStruct hwApi;
  unsigned            pios;

  [...]
  hwApi.size = 2;          // space for two functions in the struct
  pfe_get_hwapi_func_ptr(&hwApi);

  hwApi.writePios(0x55);   // write something to the pios
  pios = hwApi.readPios(); // read something from the pios
  [...]
}
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | V1.00 | n/a | n/a |

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 after your handler performs it's return.

The stack size must be at least 400 bytes.

Do not use any floating point arithmetic in your interrupt service routine.

This function is only for compatibility to older version of the hardware API. It is now deprecated. Please use interrupt **0xA2** function 0x84 and interrupt **0xA1** function 0x84 instead.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | n/a |

Top of list
Index page

---

## Interrupt 0xA1 service 0x80: HAL: Read Data Bus

Read from specified address.

## Parameters

*AH*
Must be 0x80.

*DI*
Address

*BX*
wAND

*CX*
wXOR

## SC1x Return Value

8 Bit data in AX, AX = (databus & wAND) ^ wXOR

## SC1x3 Return Value

16 Bit data in AX, AX = (databus & wAND) ^ wXOR

## Comments

& = bit wise AND
^ = bit wise XOR

The result is combined with wAND and wXOR parameters. To read the data bus without change, set wAND=0xFFFF and wXOR=0x0000.

## Related Topics

Block **Read** Data Bus
**Write** Data Bus

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

**Top of list**
**Index page**

---

## Interrupt 0xA1 service 0x81: HAL: Write Data Bus

Write to specified address.

## Parameters

*AH*
Must be 0x81.

*DI*
Address

*BX*
wAND

*CX*
wXOR

## SC1x Parameters

*DL*
8 bit data

## SC1x3 Parameters

*DX*
16 bit data

## Return Value

none

## Comments

& = bit wise AND
^ = bit wise XOR

The provided parameters are combined as follows to form the output byte value:

```
output value = (data & wAND) ^ wXOR
```

To write the value in DX to the address without modification, set wAND=0xFFFF and wXOR=0x0000.

## Related Topics

Block **Write** Data Bus
**Read** Data Bus

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

**Top of list**
**Index page**

---

## Interrupt 0xA1 service 0x82: HAL: Read Programmable I/O Pins

Read the programmable I/O pins.

## Parameters

*AH*
Must be 0x82.

## SC1x Parameters

*BX*
wAND

*CX*
wXOR

## SC1x3 Parameters

*BX*
wAND for PIOs 0-15

*CX*
wXOR for PIOs 0-15

*SI*
wAND for PIOs 16-24, 26-31

*DI*
wXOR for PIOs 16-24, 26-31

## SC23 Parameters

*BX*
wAND for PIOs 0-2, 9-13

*CX*
wXOR for PIOs 0-2, 9-13

*SI*
wAND for PIOs 18-23, 27-28, 31

*DI*
wXOR for PIOs 18-23, 27-28, 31

## SC1x Return Value

AX = (PIO[0..13] & wAND) ^ wXOR.

## SC1x3 Return Value

AX = (PIO[0..15] & wAND) ^ wXOR
DX = (PIO[16..24, 26..31] & wAND) ^ wXOR.

## SC23 Return Value

AX = (PIO[0..2, 9..13] & wAND) ^ wXOR
DX = (PIO[18..23, 27..28, 31] & wAND) ^ wXOR.

## Comments

& = bit wise AND
^ = bit wise XOR

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.

## Related Topics

Read **Specific** I/O Pin
**Write** Programmable I/O Pins
PFE: **Enable** Programmable I/O Pins

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA1 service 0x83: HAL: Write Programmable I/O Pins

Write to the programmable I/O pins.

## Parameters

*AH*
Must be 0x83.

## SC1x Parameters

*BX*
wAND

*CX*
wXOR

*DX*
data applied to PIO[0..13]
where DX bit 0 maps to PIO[0]
and DX bit 13 maps to PIO[13]

## SC1x3 Parameters

*BX*
wAND for PIO[0..15]

*CX*
wXOR for PIO[0..15]

*DX*
data applied to PIO[0..15]

where DX bit 0 maps to PIO[0]
and DX bit 15 maps to PIO[15]

## SC23 Parameters

*BX*

wAND for PIO[0..2, 9..13]

*CX*

wXOR for PIO[0..2, 9..13]

*DX*

data applied to PIO[0..2, 9..13]
where DX bit 0 maps to PIO[0]
and DX bit 13 maps to PIO[13]

## Return Value

none

## Comments

& = bit wise AND
^ = bit wise XOR

Before the value is written, the value is combined with the wAND and wXOR parameters as:

```
PIO[] = (data & wAND) ^ 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.

## Related Topics

Write **Specific** I/O Pin
**Read** Programmable I/O Pins
PFE: **Enable** Programmable I/O Pins

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.07 | V1.00 |

**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.

*DH*

Bit mask for the ISR type:

BIT0..6 must be 0

BIT7: 0=normal ISR, 1=RTX ISR (RTX ISR allow RTX Calls, e.g. wake a task), please notice the comment below!

*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

## SC1x Parameters

*DL*

HAL interrupt number from following list:

0 = INT0 (external)
1 = *(*) Network controller (internal)*
2 = INT2 (external)
3 = INT3 (external)
4 = INT4 (external) (no RTX ISR allowed)
5 = INT5 (external) / DMA Interrupt Channel 0 (if DMA is used)
6 = INT6 (external) / DMA Interrupt 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 = *(*) DMA Interrupt Channel 2 (internal, not on SC12)*
14 = *(*) DMA Interrupt Channel 3 (internal, not on SC12)*
15 = NMI Powerfail Interrupt (internal/external)

(*) = used internally by @CHIP-RTOS, not available for user interrupt service functions

## SC1x3 Parameters

*DL*

HAL interrupt number from following list:

0 = *(*) Network controller 0 (internal)*
1 = INT1 (external)
2 = SPI controller (internal)
3 = INT3 (external) / *(*) USB controller (internal)*
4 = *(*) Serial port 2 (internal)*
5 = DMA Interrupt Channel 0
6 = DMA Interrupt Channel 1
7 = INT5 (external) / *(*) Serial port 3 (internal)*
8 = Timer0 (internal)
9 = Timer1 (internal)
10 = *(*) Timer 1ms (internal)*
11 = *(*) Serial port 0 (internal)*
12 = *(*) Serial port 1 (internal)*
13 = DMA Interrupt Channel 2 (no RTX ISR allowed)
14 = DMA Interrupt Channel 3
15 = PFI Powerfail Interrupt (internal/external)
16 = *(*) CAN0 controller (internal) / (*) CAN1 controller (internal)*

17 = I2C controller (internal)
18 = Network controller 1 (internal)

(*) = used internally by @CHIP-RTOS, not available for user interrupt service functions

## SC23 Parameters

*DL*

HAL interrupt number from following list:

0 = *(*) Network controller 0 (internal)*
1 = INT1 (external)
2 = SPI controller (internal)
3 = INT3 (external) / *(*) USB controller (internal)*
4 = *(*) Serial port 2 (internal)*
5 = *(*) DMA Interrupt Channel 0*
6 = *(*) DMA Interrupt Channel 1*
7 = INT5 (external) / *(*) Serial port 3 (internal)*
8 = Timer0 (internal)
9 = Timer1 (internal)
10 = *(*) Timer 1ms (internal)*
11 = *(*) Serial port 0 (internal)*
12 = *(*) Serial port 1 (internal)*
13 = *(*) DMA Interrupt Channel 2*
14 = *(*) DMA Interrupt Channel 3*
15 = PFI Powerfail Interrupt (internal/external)
16 = *(*) CAN0 controller (internal) / (*) CAN1 controller (internal)*
17 = I2C controller (internal)
18 = *(*) Network controller 1 (internal)*

(*) = used internally by @CHIP-RTOS, not available for user interrupt service functions

## 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 can be declared in either of the following forms:

```
void huge My_ISR(void) ;     // More efficient form

void interrupt My_ISR(void) ;     // Tolerated form
```

The more efficient `huge` procedures are recommended. For assembly language ISR implementations, a far RET is recommended and a IRET on exit is tolerated. The user ISR function must preserve only the DS and BP registers, so there is no requirement for the full register save/restore provided by the `interrupt` declarations.

Any required EOI signal is issued by the system ISR which calls your user ISR function. This EOI is issued after your function returns.

If you install a RTX ISR you can use the following RTX calls in your ISR:

**RTX_RESTART_TASK**
**RTX_WAKEUP_TASK**
**RTX_SIGNAL_SEM**
**RTX_SIGNAL_EVENTS**
**RTX_SEND_MSG**
**RTX_START_TIMER**

RTX_STOP_TIMER
RTX_REMOVE_TIMER

**Important Notes:**
A RTX ISR is slower than a normal ISR.

## SC1x Comments

RTX ISR are not recommended for INT5 or INT6 if DMA is used by the Fossil serial ports interface, because the slower RTX ISR could result in UART receiver character loss.

If you are using a RTX ISR for timer0, timer1, INT5 or INT6 there must not exist a normal ISR on the system which enables the interrupts during its execution!

Do not install a RTX ISR for the powerfail interrupt!

**Also important** : The NMI function of the multifunction pin 17 (RESET/NMI/LINK_LED) of the IPC@CHIP® SC11/SC12/SC13 is for power fail purposes only. It is not possible to use NMI as a "normal" interrupt pin like INT0 for generating interrupts. It can only be used as described in the IPC@CHIP® hardware documentation.

## SC1x3 Comments

INT1 and network controller 1 share the same interrupt. Due to this fact, the installed user interrupt service routines for INT1 and network controller 1 must be of same type. If the network controller 1 is used both ISRs must be of type RTX ISR.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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.

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.

If "use internal clock" is selected the associated TMRIN pin serves as a gate. A "high" on the TMRIN pin keeps the timer counting. A "low" holds the timer value.

## SC1x Comments

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).

## SC1x3/SC23 Comments

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.
If the clock divider value is set to 0000h, the timer will not count from 0000h to FFFFh like on SC12 and SC13.

## Related Topics

**Initialize** Timer Settings Ext
HAL **Start** Timer function
**Read** Timer Count
**Write** Timer Count

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA1 service 0x8A: HAL: Get Frequencies

Get the system frequencies.

### Parameters

*AH*

Must be 0x8A.

## SC1x Parameters

*AL*

Which frequency to get:

0 = Return processor frequency
1 = Return maximum TMROUT square wave frequency
2 = Return maximum baud rate
3 = Return timer base freqeuncy / PWD timer frequency

## SC1x3/SC23 Parameters

*AL*

Which frequency to get:

0 = Return processor frequency
1 = Return maximum TMROUT square wave frequency
2 = Return maximum baud rate
3 = Return timer base frequency

## Return Value

DX:AX frequency [Hz]

## Comments

Use the timer base frequency to compute the correct timer clock divider value, where:

```
Timer frequency = timer base frequency / clock divider
```

Use the maximum TMROUT square wave frequency to compute the correct timer clock divider value for the TMROUT, where:

```
TMROUT frequency = maximum TMROUT square wave frequency / clock divider
```

Use the maximum baud rate to compute the correct value for the processor specific baud rate initialize function (See Fossil *Extended line control initialization* **function**).

```
Baud rate = maximum baud rate / baud rate divider
```

## Related Topics

HAL **Initialize** Timer Settings

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.05 | V1.00 |

**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. If you want an alternate duty cycle waveform, compute it with the following formula:

```
Output frequency = Maximum TMROUT square wave frequency * 2
            / (divider high level + divider low level)
```

The `divider high level` is the divider set by the Timer Initialization **function**. The `divider low level` is the alternate clock divider passed to this function in the CX register.

## Related Topics

HAL **Initialize** Timer Settings

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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.

## SC1x Parameters

*AL*
IPC@CHIP® PIO No [0..13]

## SC1x3 Parameters

*AL*
IPC@CHIP® PIO No [0..24, 26..31].

## SC23 Parameters

*AL*
IPC@CHIP® PIO No [0..2, 9..13, 18..23, 27..28, 31].

## 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## 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.

*DX*
DX = 0 ==> set PIO to low
DX non-zero ==> set PIO to high

## SC1x Parameters

*AL*
IPC@CHIP® PIO No [0..13]

## SC1x3 Parameters

*AL*
IPC@CHIP® PIO No [0..24, 26..31]

## SC23 Parameters

*AL*
IPC@CHIP® PIO No [0..2, 9..13, 18..23, 27..28, 31]

## Return Value

none

## Related Topics

**Read** Specific I/O Pin
**Write** Programmable I/O Pins
PFE: **Enable** Programmable I/O Pins

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## 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 your own interrupt service routines without the HAL function **Install Interrupt Service Routine**.

This function is especially needed for generating an EOI for INT0 when using cascaded mode of the interrupt controller with INT0/INTA .

## Related Topics

**Enable INT0/INTA** cascade mode

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | n/a | n/a |

**Top of list**
**Index page**

---

## Interrupt 0xA1 service 0x8F: HAL: Initialize Timer Settings Ext

Some more useful timer settings (extends the Initialize Timer Settings **function**)

## Parameters

*AH*

Must be 0x8F.

*AL*

Timer
0=Timer0 / 1=Timer1

*DX*

Mode
Bit 0..2: must be 0
Bit 3: 0=disable prescale t2 / 1=enable prescale t2
Bit 4: 0=disable retrigger / 1=enable retrigger
Bit 5..15: must be 0

## Return Value

none

## Comments

The Initialize Timer Settings **function** must be called prior to this function!

**Prescale T2:**

If the Prescale feature is enabled, the timer (specified in AL) will use Timer2 output for its timer base (clock input). If the **RTI rate**is not changed by the user it will provide a 1000 Hz timer clock rate. (Timer2 is used internally by the @CHIP-RTOS as the Real-Time Interrupt (RTI), which defaults to a one millisecond interval timer.) If this Prescale bit is disabled, the timer base will instead be the frequency reported by the **Get Frequencies**API function.

## SC1x Comments

**Retrigger:**

If the retrigger setting is enabled, a 0 to 1 edge transition on TMRIN0 (or TMRIN1) resets the timer count. When retrigger is disabled (DX bit 4 set to 0), a High input on TMRIN0 (or TMRIN1) enables counting and a Low input holds the timer value.

## SC1x3/SC23 Comments

**Retrigger:**

If the retrigger setting is enabled, a 0 to 1 edge transition on TMRIN0 (or TMRIN1) resets the timer count. If the input remains High, the timer starts counting on the following clock transitions. If the input goes again Low, the timer stops counting. When retrigger is disabled (DX bit 4 set to 0), a High input on TMRIN0 (or TMRIN1) enables counting and a Low input holds the timer value.

## Related Topics

**Initialize** Timer Settings
HAL **Start** Timer function
**Read** Timer Count
**Write** Timer Count

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 @CHIP-RTOS (default)

## Return Value

AX=mode

## SC1x 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 @CHIP-RTOS mode, the @CHIP-RTOS 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.

## SC1x3/SC23 Comments

The watchdog timeout period is 699 ms.

If you select the *User Program* mode, you must call the HAL **Refresh** Watchdog function 0x91 at least every 699 ms to prevent the watchdog from resetting the system. In @CHIP-RTOS mode, the @CHIP-RTOS 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

**Refresh** Watchdog Function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# Interrupt 0xA1 service 0x91: HAL: Refresh Watchdog

Strobe the hardware watchdog to reset its timeout period.

## Parameters

*AH*
Must be 0x91.

## Return Value

none

## SC1x 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.

## SC1x3/SC23 Comments

If the watchdog is in *User Program* mode, this function must be called at least every 699 ms to prevent a CPU reset due to watchdog timeout.

## Related Topics

Get/Set **Watchdog** Mode

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA1 service 0x92: HAL: Mask/Unmask Int

Mask or unmask an external Interrupt Request

## Parameters

*AH*

Must be 0x92.

*AL*

1=Mask, 0=Unmask

## SC1x Parameters

*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) / DMA Interrupt Channel 0 (if DMA is used)
6 = INT6 (external) / DMA Interrupt 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 = DMA Interrupt Channel 2 (internal, not on SC12)
14 = DMA Interrupt Channel 3 (internal, not on SC12)
15 = NMI (Not maskable!)

## SC1x3/SC23 Parameters

*DX*

HAL interrupt number from following list:

0 = Network controller 0 (internal)
1 = INT1 (external) / Network controller 1 (internal)
2 = SPI controller (internal) / I2C controller (internal)
3 = INT3 (external) / USB controller (internal)
4 = Serial port 2 (internal)
5 = DMA Interrupt Channel 0
6 = DMA Interrupt Channel 1
7 = INT5 (external) / Serial port 3 (internal)
8 = Timer0 (internal)

9 = Timer1 (internal)
10 = Timer 1ms (internal)
11 = Serial port 0 (internal)
12 = Serial port 1 (internal)
13 = DMA Interrupt Channel 2
14 = DMA Interrupt Channel 3
15 = NMI (Not maskable)
16 = CAN0 controller (internal) / CAN1 controller (internal)

## Return Value

none

## Comments

**CAUTION:**
Masking any of the three timer interrupts will suspend the @CHIP-RTOS 1000 Hz real-time interrupt, essential for system operation. Consequently this mask period should be very brief, if used at all.

## SC1x Comments

Some interrupts share the same mask bit. If you mask one of them, the other interrupts which are assigned to the same bit are also masked. Here are the groups which are masked together:

Timer0, Timer1, Timer 1ms
DMA0, INT5
DMA1, INT6

## SC1x3/SC23 Comments

Some interrupts share the same mask bit. If you mask one of them, the other interrupts which are assigned to the same bit are also masked. Here are the groups which are masked together:

Timer0, Timer1, Timer 1ms
INT1 / Network controller 1
SPI controller / I2C controller
INT3 / USB controller
INT5 / Serial port 2
CAN0 controller / CAN1 controller

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA1 service 0x93: HAL: Write Programmable I/O Pins (Extended)

Write to the programmable I/O pins (PIOs 16-31).

## Parameters

*AH*

Must be 0x93.

## SC1x3 Parameters

*DX*

data applied to PIO[16..24, 26..31]
where DX bit 0 maps to PIO[16]
and DX bit 15 maps to PIO[31]

*BX*

wAND for PIO[16..24, 26..31]

*CX*

wXOR for PIO[16..24, 26..31]

## SC23 Parameters

*DX*

data applied to PIO[18..23, 27..28, 31]
where DX bit 2 maps to PIO[18]
and DX bit 15 maps to PIO[31]

*BX*

wAND for PIO[18..23, 27..28, 31]

*CX*

wXOR for PIO[18..23, 27..28, 31]

## Return Value

none

## Comments

& = bit wise AND
^ = bit wise XOR

Before the value is written, the value is combined with the wAND and wXOR parameters as:

```
PIO[] = (data & wAND) ^ 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.

## Related Topics

Write **Specific** I/O Pin
**Read** Programmable I/O Pins
PFE: **Enable** Programmable I/O Pins

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## 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 != 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | n/a |

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | n/a |

**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 when the external DRQ pins is activated.

## Parameters

*AH*
Must be 0xB0

*CX*
Number of bytes which has 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
Bit 11: 1=Word Transfer / 0=Byte Transfer
Bit1 and Bit2 can't be set at the same time.

## SC1x Parameters

*AL*

DMA channel:
0 = DRQ0
1 = DRQ1

*BX:SI*

Pointer to unsigned long (20-Bit physical) source address

*ES:DI*

Pointer to unsigned long (20-Bit physical) destination address

## SC1x3 Parameters

*AL*

DMA channel:
0 = DRQ0
1 = DRQ1
2 = DRQ2
3 = DRQ3

*BX:SI*

Pointer to unsigned long (24-Bit physical) source address

*ES:DI*

Pointer to unsigned long (24-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 is activated the DMA transfer will be started. The number of bytes which will be transferred has to be specified in the CX register. Before calling this function you have to enable the DRQ pin for this channel.

## Related Topics

Enable external DRQ Pin
HAL Stop DMA Transfer

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | n/a |

Top of list
Index page

---

## Interrupt 0xA1 service 0xB1: HAL: Stop DMA Transfer

Disables the DMA controller. A running DMA transfer will be halted.

## Parameters

*AH*
Must be 0xB1

## SC1x Parameters

*AL*
DMA channel:
0 = DRQ0
1 = DRQ1

## SC1x3 Parameters

*AL*
DMA channel:
0 = DRQ0
1 = DRQ1
2 = DRQ2
3 = DRQ3

## 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 reading the current DMA values (using function Get DMA Info) and starting the DMA Transfer again with the read values (using function Start DMA Mode)

## Related Topics

Start DMA Transfer

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | n/a |

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

## SC1x Parameters

*AL*
DMA channel:
0 = DRQ0
1 = DRQ1

*BX:SI*
Output Parameter: Pointer to unsigned long where this function will write the (20-Bit physical) source address

*ES:DI*
Output Parameter: Pointer to unsigned long where this function will write the (20-Bit physical) destination address

## SC1x3 Parameters

*AL*
DMA channel:
0 = DRQ0
1 = DRQ1
2 = DRQ2
3 = DRQ3

*BX:SI*
Output Parameter: Pointer to unsigned long where this function will write the (24-Bit physical) source address

*ES:DI*
Output Parameter: Pointer to unsigned long where this function will write the (24-Bit physical) destination address

## 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 yet 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
Bit 11: 1=Word Transfer / 0=Byte Transfer

[BX:SI] = contains unsigned long (physical) DMA source address
[ES:DI] = contains unsigned long (physical) DMA destination address

## Comments

This function returns the status of the DMA channel in AX.

## Related Topics

**Start** DMA Transfer

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | n/a |

**Top of list**
**Index page**

---

## Interrupt 0xA1 service 0xC0: HAL: Initialize/Restore Non-Volatile Data

Initialize/Restore non-volatile data. Tell the @CHIP-RTOS where your variables are located, which should be saved and reload their saved values, if available.

**SC1x:**
The non-volatile (remanent) data is stored in `A:\rema.bin` file.

**SC1x3/SC23:**
The non-volatile (remanent) data is stored in a reserved flash memory area. The maximum size is approx. 8 KB.

## 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;      // Address offset
        unsigned size;      // Number of bytes
        unsigned maxsize;   // Obsolete, set to 0
        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, Error

## 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.)

The number of entries in the x array of REMOP_ENTRY data structures can be defined by the user. This number must be specified in the `entries` field of the _REMOP data structure.

All data saved / restored must reside in the same segment, specified by the `segment` field in _REMOP. The individual data item locations in this segment are then listed in the `offs` fields of the REMOP_ENTRY structure array.

## Related Topics

Save Non-Volatile Data

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xA1 service 0xC1: HAL: Save Non-Volatile Data

This function saves your non-volatile data listed in the _REMOP structure registered with HAL function 0xC0.

**SC1x:**
The data is stored in `A:\rema.bin` file.

**SC1x3/SC23:**

The data is stored in a reserved flash memory area.

## Parameters

*AH*
Must be 0xC1

## Return Value

none

## Comments

Call this function on exit from your program and in your powerfail (NMI, Non-Maskable Interrupt) handler. Your hardware around the IPC@CHIP® must support the Pfail signal, so that the IPC@CHIP® can generate an powerfail interrupt sufficiently in advance of power loss to the CPU.

## SC12 Comments

**Reminder** : The SC12 evalution board DK40 does not support the Pfail signal.

## Related Topics

**Initialize/Restore** Non-Volatile Data
Install **Interrupt** Service Routine

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xA1 service 0xC3: HAL: Install PowerfailDone callback

Install a callback for the event "Powerfail is Done". The callback is executed after all Powerfail actions are finished. This means that the user non-volatile data has been written and the file sytem closed.

The user can toggle a PIO inside this callback and measure the time from Powerfail (PFI/NMI) to this event.

The time to close the file system depends on the number of open files and on which drive these files are located (speed of the drive).

## Parameters

*AH*

Must be 0xC3

*ES:DI*

Pointer to callback function which should be of type:

```
typedef void huge (far *InterruptHandler)(void);
```

## Return Value

none

## Related Topics

**Initialize/Restore** Non-Volatile Data

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.21 | V1.21 | V1.21 | V1.10 | V1.00 |

**Top of list**
**Index page**

# FOSSIL

Here is the API definition for access to the serial ports.

The @Chip-RTOS offers the Fossil API for serial port communication. The Fossil standard uses software interrupt 0x14. These functions provide access to the @Chip-RTOS internal serial port driver for receiving and sending data via the serial ports.

Here is a short description how the @Chip-RTOS internal serial port driver operates:

Each serial port has two software buffers (queues), one for data pending transmission and one for storing received data bytes. The default size of each queue is 1024 bytes. The size of these transmit and receive queues are configurable via **CHIP.INI** entries.

Transfers between these software queues and the serial port hardware are carried out by either Direct Memory Access (DMA) hardware or by hardware interrupt (IRQ) driven software. By default, the available DMA machines are applied to the serial port receivers. This configuration reduces the likelihood of character loss at the receiver. This leaves the serial port transmitters operating with hardware interrupt (IRQ) driven software. Alternately, receivers can be configured for interrupt driven mode (IRQ receive mode) with the **CHIP.INI** options, which frees up the DMA device for other usage. The DMA can be applied to transmitters with the SERIAL **SEND_DMA** option.

The serial port hardware issues a signal to either software (IRQ hardware interrupt) or to the DMA when ever the serial port transmit register is empty or a receiver byte is ready. This signal initiates the next byte transfer between the appropriate software queue and the hardware register, in or out. (This discussion is slightly over simplified. The driver actually uses an additional intermediate RAM buffer for DMA transfers.)

Note that the serial port hardware send/receive buffers are only one byte (four bytes) deep on the SC1x (SC1x3/SC23). So interrupt driven receivers (as opposed to DMA driven) can easily lose characters, particularly at higher baud rates.

The serial ports may also be referred to as a UART (abbreviation for "Universal Asynchronous Receiver/Transmitter").

For some useful comments see **Programming notes**

## Fossil Interrupt 0x14 Services

Service is selected by index in AH register.

- **0x00: Set baud rate**
- **0x01: Put byte in output buffer, wait if needed.**
- **0x02: Get a byte from the serial port, wait if none available.**

- 0x03: Status request
- 0x04: Initialize fossil driver
- 0x05: Deinitialize fossil driver
- 0x08: Flush output buffer waiting until done.
- 0x09: Purge output buffer.
- 0x0A: Purge receive buffer.
- 0x0B: Transmit byte, do not wait.
- 0x0C: Peek if next byte is available.
- 0x0F: Enable/disable flow control.
- 0x18: Read block of data
- 0x19: Write a block of data
- 0x1B: Get driver info
- 0x1E: Extended set baud rate
- 0x80: Enable/Disable RS485 mode
- 0x81: Extended line control initialization
- 0x82: Select RS485 pin
- 0x83: Send break
- 0x84: Enable/disable UART receiver
- 0x85: Enable/disable UART transmitter
- 0x86: Force RTS#
- 0x87: Read CTS#
- 0xA0: Get number of Tx bytes in UART
- 0xA1: Install a Fossil User Callback Function
- 0xA2: Register an external device driver port
- 0xA3: Unregister an external device driver port


---

## Interrupt 0x14 service 0x00: Set baud rate

Set the baud rate of the serial port

## Parameters

*AH*

0x00

*AL*

Configuration parameter:
This bitmap field specifies the communication parameters.

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

Bits 1-0: Word length
10 7 bits
11 8 bits

*DX*

Port number:
0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

AH: Status code (see service **0x03**)

## Comments

For higher baud rates use service **0x81** "Extended line control initialization"

## SC1x Comments

Two stop bits are only available if no parity is set.

## SC1x3/SC23 Comments

Two stop bits are only available if no parity is set or word length is equal to 7 bits.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

AH: Status code (see service **0x03**)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

AL: The byte received

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## 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.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

## Interrupt 0x14 service 0x04: Initialize fossil driver

Initialize the fossil driver for specified port.

## Parameters

*AH*

0x04

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)

1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## 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.

If the DMA mode (send or receive mode) is enabled in **CHIP.INI**, the following port settings are not allowed:

1. 8N2 (this limitation applies only on SC1x systems)
2. 8S1
3. 8M1

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x14 service 0x05: Deinitialize fossil driver

Deinitialize the fossil driver for specified port.

## Parameters

*AH*

0x05

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## 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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x14 service 0x09: Purge output buffer.

Remove all data from the output buffer.

## Parameters

*AH*

0x09

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)

2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x14 service 0x0B: **Transmit byte, do not wait.**

Place a byte into 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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

AX=0 if byte was not accepted (no space in buffer)
AX=1 if byte was placed in buffer

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

AX=0xFFFF if no byte was available
AH=0x00 and AL=next byte, if a byte was available.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## 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.

XON/XOFF mode is also available if the **DMA** mode for the 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 now provide this mode because GSM modems (any??) support only XON/XOFF flow ctrl.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

*ES:DI*

Pointer to user buffer.

## Return Value

AX= Number of bytes transferred.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x14 service 0x19: Write a block of data

Write a block of data to the serial port output buffer.

## Parameters

*AH*

0x19

*CX*

Maximum number of bytes to transfer.

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

*ES:DI*

Pointer to user buffer.

## Return Value

AX= Number of bytes actually transferred.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

*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 provided for compatibility with older Fossil applications.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x14 service 0x1E: Extended set baud rate

Set the baud rate of the serial port

## Parameters

*AH*

0x1E

*BH*

Parity
- 00h None
- 01h Odd
- 02h Even
- 03h Mark
- 04h Space

*BL*

Stop bits
- 00h: 1 Stop bit
- 01h: 2 Stop bits

*CH*

Word length
- 02h: 7 bits
- 03h: 8 bits

*CL*

Baud rate
- 00h: 110

01h: 150
02h: 300
03h: 600
04h: 1200
05h: 2400
06h: 4800
07h: 9600
08h: 19200
80h: 28800
81h: 38400
82h: 57600
83h: 76800
84h: 115200

*DX*

Port number:
0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

AH: Status code (see service **0x03**)

## SC1x Comments

Two stop bits are only available if no parity is set.

## SC1x3/SC23 Comments

Two stop bits are only available if no parity is set or word length is equal to 7 bits.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Comments

By default the RTS signals (pins) are used to enable/disable the respective serial port transmitter. (TxEnable)

## SC1x Comments

Note that RS485 is not available with **serial send DMA**!

## SC1x3/SC23 Comments

Note that RS485 is only available with **serial send DMA** when the TxEnable hardware function is used! See **Select RS485 pin** function.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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

*CX*

Baud rate divider
(for maximum baud rate see HAL function **0x8A**)

*DX*

Port number:
0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Comments

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.

If the DMA mode (send or receive mode) is enabled in **CHIP.INI**, the following port settings are not allowed:

1. 8N2 (this limitation applies only on SC1x systems)
2. 8S1
3. 8M1

## SC1x Comments

Two stop bits are only available if no parity is set.

## SC1x3/SC23 Comments

Two stop bits are only available if no parity is set or word length is equal to 7 bits.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0x14 service 0x82: Select RS485 pin

Select the RS485 TxEnable pin

### Parameters

*AH*

0x82

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

### SC1x Parameters

*AL*

No of PIO pin [0..13]

### SC1x3 Parameters

*AL*

No of PIO pin [0..24, 26..31]
Use -1 to select the RTS pin (hardware functionality / default)

### Return Value

none

### Comments

By default the RTS signals (pins) are used to enable/disable the respective transmitter. (TxEnable) This function lets you select any PIO as TxEnable, but not all make sense. To change the default, call this function before you call the RS485 **Enable** function.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0x14 service 0x83: Send break

Send long or short break

## Parameters

*AH*

0x83

*AL*

1: long break (2,5 frames)
2: short break (1 frame)
3: extra long break (3 frames)

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Comments

A short break is a continuous Low on the TXD output for a duration of more than one frame transmission time `M`, where:

```
M = startbit + data bits (+ parity bit) + stop bit
```

A long break is a continuous Low on the TXD output for a duration of more than two frame transmission times plus the transmission time for three additional bits (2M+3).

A extra long break is a continuous Low on the TXD output for a duration of more than three frame transmission times.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x14 service 0x84: Enable/disable UART receiver

Enable/Disable UART receiver

## Parameters

*AH*

0x84

*AL*

0: disable receiver 1: enable receiver

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Comments

This function is useful when using the SM35 as RS485 adapter. Use this function to disable the receiver before transmitting data. Wait until all data is sent and then reenable the receiver. This prevents receiving your own transmitted data.

By default the receiver is enabled.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x14 service 0x85: Enable/disable UART transmitter

Enable/Disable UART transmitter

## Parameters

*AH*

0x85

*AL*

0: disable transmitter 1: enable transmitter

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Comments

By default the transmitter is enabled.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x14 service 0x86: Force RTS#

Force RTS# to low, high or reset to normal activity

## Parameters

*AH*

0x86

*AL*

Mode:

0: force RTS to low
1: force RTS to high
2: reset to normal activity

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

none

## Comments

RTS/CTS flow control must be enabled to use this function.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0x14 service 0x87: Read CTS#

Read level on CTS# pin

## Parameters

*AH*

0x87

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

AL=0, CTS# is low
AL=1, CTS# is high

## Comments

RTS/CTS flow control must be enabled to use this function.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0x14 service 0xA0: Get number of Tx bytes in UART

Returns the number of bytes which are currently in the UART transmitter hardware.

## Parameters

*AH*

0xA0

*DX*

Port number:

0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

## Return Value

AX= Number of bytes in the UART transmitter hardware
-1 if UART transmitter is disabled (since SC1x version V1.20 and SC1x3 version V1.05)

## Comments

With this function you can check how many bytes are currently in the UART. This could be necessary to know if your communication pauses because of a handshake problem.

The GetDriverInfo API (**0x1B**) reports the number of bytes remaining in the serial port driver's software send queue.

This function reports the number of bytes currently in the UART transmit hardware. By adding this function's return value to the software transmit buffer byte count reported by the GetDriverInfo API, you can determine the total number of transmit data bytes still pending output.

The maximum count returned here will be 2 bytes, accounting for the UART's transmit shift register (1 byte) and transmit holding register (1 byte).

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x14 service 0xA1: Install a Fossil User Callback Function

The user callback function will be called when a Fossil serial port event occurs.

## Parameters

*AH*
0xA1

*DX*
Port number:
0 for the EXT (COM0) port (SC1x, SC1x3)
1 for the COM (COM1) port (SC1x, SC1x3, SC23)
2 for the COM2 port (SC1x3, SC23)
3 for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see service **0xA2**).

*ES:DI*

Pointer to the User Callback function

## Comments

This service installs a user callback function for the specified serial port. The callback function must be very short! Long fossil callback functions could lead to character losing. To use the fossil callback functions you have to switch the corresponding serial port into the IRQ Mode (see **CHIP.INI**). In DMA Mode the callback functions do not work!

The callback function must be declared like the function below:

```
fossil_event_t far *(huge my_fossil_callback)(
    fossil_event_t far *e )
```

The `fossil_event_t` structure e contains the `event` which has occurred.
Following events are possible:

```
#define FE_DATA_AVAIL      0x01 // New data received
#define FE_READY_FOR_SEND  0x02 // --- currently not supported ---
#define FE_DATA_AVAIL9     0x03 // New data received with 9. data bit set
#define FE_ERROR_DETECTED  0x10 // Data contains status information
```

Newly received `data_length` bytes can be found at location referenced by `data` pointer in structure `e`.

In case of event `FE_ERROR_DETECTED` the first two bytes of the data buffer should be interpreted as an unsigned short int (16 bit) that contains status information. The third byte will hold the received char. The following bit positions are defined in the status information:

`Bit3=1` - Parity Error Detected
`Bit5=1` - Framing Error Detected
`Bit9=1` - Short Break Detected
`Bit10=1` - Long Break Detected

The callback function can also return a `fossil_event_t` structure. The returned structure contains an action `event` which the Fossil serial port driver will respond to as follows:

```
#define FE_IGNORE_DATA    0x01 // Don't copy the received data into the receive queue.
#define FE_DATA_TO_SEND   0x02 // --- currently not supported ---
```

If the callback does not return any event, the return value must be a null pointer (=0L).

The definition of the `fossil_event_t` structure is as follows:

```
typedef struct
{
    int size;             // size of the structure
    int port;             // serial port (e.g. 0=EXT, 1=COM)
    int event;            // event (see above)
    void far *data;       // data pointer
    unsigned data_length; // data length
} fossil_event_t;
```

The `size` element should be set to `sizeof(fossil_event_t)`.

Note that only one user callback function can be installed per port. To uninstall a user callback function, simply call this service again with another function pointer or with a NULL pointer.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.17 | V1.17 |

Top of list
Index page

## Interrupt 0x14 service 0xA2: Register an external device driver port

This function registers an additional serial port with the RTOS, which is handled by an external device driver.

### Parameters

*AH*
0xA2

*DX*
Designated port number; If a special port number is to be assigned to the external device driver it can be specified here. If the external device driver doesn't care about its port number, a value < 0 should be handed. The RTOS will assign the next free port number in this case.

*ES:DI*
Pointer to the external device driver's callback function. The signature of this callback function must be as follows:

```
unsigned huge callback(unsigned ax, unsigned bx, unsigned cx,
                       unsigned dx, unsigned es, unsigned di);
```

### Return Value

AX == Assigned port number: The external port has been successfully registered.
AX == -1: The designated port number is already in use.
AX == -2: The maximum number of external ports is already registered.

### Comments

After an external port has been installed via this function the RTOS will forward all fossil software interrupts regarding the corresponding port to the callback function of the device driver program. It is the driver program's duty to parse the parameters.

### Developer Notes

This API service is intended to be used by the RTOS C-Library exclusively.
It should not be used within user applications.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.17 | V1.17 |

Top of list
Index page

## Interrupt 0x14 service 0xA3: Unregister an external device driver port

This function unregisters an additional serial port.

## Parameters

*AH*

0xA3

*DX*

Port number of the external device driver port

## Return Value

AX == 0: The external port has been successfully unregistered.
AX == -1: Illegal port number

## Developer Notes

This API service is intended to be used by the RTOS C-Library exclusively.
It should not be used within user applications.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.17 | V1.17 |

**Top of list**
**Index page**

---

End of document

---




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS - DOS Interface Documentation



## DOS

DOS API Here are the DOS interface definitions. DOS uses interrupt 0x21 with a service number in the high order byte of the AX register (AH).

For some useful comments see also under **Programming notes**

### DOS Interrupt 0x21 Services

Service is selected by index in AH register. All implemented DOS services are listed here:

- **0x00: Terminate Program**
- **0x02: Output Character to standard output**
- **0x06: Direct Console Output**
- **0x07: Direct Console Input**
- **0x08: Read Keyboard**
- **0x09: Send string to standard output**
- **0x0B: Character Available Test**
- **0x0E: Set Default Drive**
- **0x19: Get Current Drive**
- **0x1A: Set Disk Transfer Area Address**
- **0x25: Set IRQ Vector**
- **0x2A: Get System Date**
- **0x2B: Set System Date**
- **0x2C: Get System Time**
- **0x2D: Set System Time**
- **0x2F: Get Disk Transfer Area Address**
- **0x30: Get DOS Version**
- **0x31: Keep Process**
- **0x35: Get IRQ Vector**
- **0x36: Get Disk Free Space**
- **0x39: Create Directory**
- **0x3A: Remove Directory**
- **0x3B: Set Current Working Directory**
- **0x3C: Create New File Handle**
- **0x3D: Open an Existing File**
- **0x3E: Close File Handle**
- **0x3F: Read from File**
- **0x40: Write to File**
- **0x41: Delete File**
- **0x42: Set Current File Position**
- **0x43: Get/Set File Attributes**

- 0x44: _IOCTL, Set/Get Device Information
- 0x47: _Get Current Working Directory
- 0x48: _Allocate Memory
- 0x49: _Free Allocated Memory
- 0x4A: _Resize Memory
- 0x4B: _EXEC
- 0x4C: _End Process
- 0x4E: _Find First File
- 0x4F: _Find Next File
- 0x50: _Debugger Support
- 0x51: _Get PSP Segment Address
- 0x56: _Change Directory Entry, Rename File
- 0x57: _Get/Set File Date and Time
- 0x58: _Get/Set memory strategy (dummy function)
- 0x62: _Get PSP Segment Address
- 0x63: _Get Leading Byte (stub)
- 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 be run simultaneously.

The system supports a maximum of 20 open 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 treats both as the same function.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x21 service 0x02: Output Character to standard output

Sends 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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
        Set **BX**register to indicate the `stdin`channel source of character,
            where: 1: EXT , 2: COM , 4: Telnet.
        Return input character in **AL**and reset CPU's zero flag
    Else

```
                Set CPU's zero flag to indicate no character available
            Endif
        Endif
```

## Comments

Output is buffered and interrupt driven. This function will return after placing output character into the transmit buffer when DL != 0xFF.

This function does not check for Ctrl-C.

## Related Topics

**stdin** configuration
**stdout** configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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.
Returns in BX the source `stdin` channel of the character, where: 1: EXT , 2: COM , 4: Telnet

## 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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 in BX the source `stdin` channel of the character, where: 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0x21 service 0x09: Send string to standard output

Sends 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x21 service 0x0E: Set Default Drive

Changes the default drive for the current task.

## Parameters

*AH*

0x0E

*DL*
New default drive (00h = A:, 01h = B:, etc)

## Return Value

AL = total number of drives on success, else -1

## Related Topics

**Get** current drive

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.17 | V1.17 |

**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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x21 service 0x1A: Set Disk Transfer Area Address

Sets address of the Disk Transfer Area (DTA) needed for `findfirst/findnext` functions.

## Parameters

*AH*

0x2A

*DS:DX*
Pointer to DTA

## Comments

This call is used within your compiler's standard C-library `findfirst()` function. Prior to using the findfirst/findnext functions a buffer area must be set with this API.

## Related Topics

DOS **Get** Disk Transfer Area address service
DOS **findfirst** service
DOS **findnext** service

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**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.

## Parameters

*AH*
0x25

*AL*
Specifies vector number.

*DS:DX*
Vector to your interrupt procedure.

## Return Value

No return value.

## Comments

You can use the following IRQ's

0x0A DMA0 / INT5
0x0B DMA1 / INT6
0x0C INT0
0x0E INT2
0x0F INT3
0x10 INT4

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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)

## Return Value

AL = 0 if date was accepted
AL = 0xFF if date was rejected by the limited error checking

## Comments

This function performs only limited error checking on the entered date.

## Related Topics

**Get** system date
**Set** system time

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0x21 service 0x2D: Set System Time

Sets the system time.

### Parameters

*AH*
0x2D

*CH*
Hour

*CL*
Minute

*DH*
Second

### Return Value

AL = 0

### Comments

This function performs no error checking on the entered time values.

### Related Topics

**Set** system date
**Get** system time

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0x21 service 0x2F: Get Disk Transfer Area Address

Gets address of the Disk Transfer Area (DTA) needed for `findfirst/findnext`.

### Parameters

*AH*
0x2F

### Return Value

Returns the address of the DTA in ES:BX

## Related Topics

DOS **Set** Disk Transfer Area address service
DOS **findfirst** service
DOS **findnext** service

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## 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 DOS version 3.00).

However, this does not mean that we have a full implementation of DOS 3.0 !

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## 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

## Comments

The DOSn @Chip-RTOS task supporting this terminating DOS program will be deleted. The memory allocated from the system by this task will remain allocated and marked as being owned by TSR (Terminate/Stay Resident).

After this calling program's task has been deleted, the program's memory image will be reduced to the size specified in DX register. Since this action is taken after the task has been retired, it is safe to delete the programs stack space as well so that the TSR will have a yet smaller memory image following the termination.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0x21 service 0x35: Get IRQ Vector

Gets 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0x21 service 0x36: Get Disk Free Space

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: number of sectors per cluster
BX: number of free clusters
CX: number of bytes per sector
DX: number of clusters per drive

## Comments

When call is successful (AX=1), free disk space can be computed from the return values as:

free disk space (bytes) = AX * BX * CX

For drives with more than 0xFFFF clusters, the BIOS **API** alternative function should be used.

Note:
In SC12 @CHIP-RTOS version 1.00, this function had a bug: Parameter 0 in DL was taken as drive A:, which should have meant the current drive.

## Related Topics

BIOS **service 0x36** Get Disk Free Space

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x21 service 0x39: Create Directory

Creates 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x21 service 0x3A: Remove Directory

Deletes 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x21 service 0x3B: Set Current Working Directory

Sets the 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x21 service 0x3C: Create New File Handle

Creates 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
Get/Set File **Date/Time**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## 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 write mode, files are always opened in a non-sharing mode.

The file system does not distinguish between file not found or invalid path. The error return value in both cases is 2.

## Related Topics

**Create** New File Handle
**Close** File Handle
Get/Set File **Attributes**
Get/Set File **Date/Time**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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
Get/Set File **Date/Time**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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
**Set** Current File Position

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## 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

## Comments

Files with read only attribute cannot be deleted.

## Related Topics

**Create** New File Handle
Get/Set File **Attributes**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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.

For C language applications, the `_dos_getfileattr()` and `_dos_setfileattr()` functions from the standard C-library may be used to access this API.

## Related Topics

**Delete** File

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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     '1' indicates 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 three 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 the @CHIP-RTOS. Control characters are not recognized as such. Function 0x44 codes 2 and 3 are not supported.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x21 service 0x47: Get Current Working Directory

Gets the 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 it's 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x21 service 0x48: Allocate Memory

Allocates memory for the calling 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

For SC1x CPU's, a paragraph is 16 bytes in length.

For SC1x3/SC23 CPU's, a paragraph is 256 bytes in length.

## Related Topics

**Free** allocated memory
**Resize** memory
Allocate **fast** memory
**Reassign** memory ownership

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x21 service 0x49: Free Allocated Memory

Releases the specified allocated memory.

## Parameters

*AH*
0x49

*ES*
Segment of the memory area to be released, allocated with function **0x48**

## Return Value

Success: Carry flag cleared
Failure: Carry flag set, AX holds the error code = 9.

## Related Topics

**Allocate** memory
**Resize** memory

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x21 service 0x4A: Resize Memory

Increases or decreases the size of a system memory block.

## Parameters

*AH*
0x4A

*BX*
Desired new size expressed in paragraphs

*ES*
Segment address of memory block (allocated by DOS SWI 0x21 **Service 0x48**)

## Return Value

Success: Carry flag cleared

Failure: Carry flag set due to not enough memory available. In this case, the size of the largest free block is returned in BX (paragraph count).

## Comments

For SC1x CPU's, a paragraph is 16 bytes in length.

For SC1x3/SC23 CPU's, a paragraph is 256 bytes in length.

Increasing a memory block's size may fail even when there is more heap memory available than requested in register BX. This can occur when the heap area following the specified memory block (ES register) has already been allocated, leaving insufficient heap space for this block to grow. Note that this API cannot move the base address of the memory block.

## Related Topics

**Allocate** memory
**Free** memory

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 use in child process (system default environment used if 0000h). |
| 02h | DWORD | Pointer to command tail to be copied into child's PSP The command tail string should be zero terminated. (Command tail pointer not used if NULL.) |
| 06h | DWORD | Pointer to first FCB (not used) |
| 0Ah | DWORD | Pointer to second FCB (not used) |
| 0Eh | DWORD | Will hold subprogram's initial SS:SP on return |
| 12h | DWORD | Will hold entry point (CS:IP) on return |

## Return Value

Success:
Carry flag cleared
AX = New program's **task**ID
BX = segment of PSP (add sizeof PSP/16 for relocation offset)

Failure:
Carry flag set
AX = 1

## Comments

Use 'Type of Load' AL = 0x00 to load and execute another program.

'Type of Load' AL = 0x01 is used to load a program without executing it. This option is available for debuggers. The new task is waiting for it's **trigger** when started in this manner.

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**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x21 service 0x4C: End Process

Terminates a DOS program.

## 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.

## Related Topics

**Delete** @CHIP-RTOS task

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 (provisional, not used!)

*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 DTA address must be set with interrupt 0x21 function **0x1A** before calling this `findfirst` function. This is handled by a compiler's C-library code. The `findfirst`/**`findnext`** sequence is handled by your compiler library so the Disk Transfer Area is therefore not described here.

## SC1x3/SC23 Comments

The alternative directory scan functions for long file names (LFN) starting with **BIOS_LFN_Findfirst** can be used to read out the long file names.

## Related Topics

Find **next** file
Fast **Findfirst** file
Long file name **Findfirst** file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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).

## Parameters

*AH*
0x4F

## Return Value

Success: Carry flag cleared, search results are in DTA
Failure: Carry flag set, AX = 0x12

## Comments

The task that calls `findnext` must be the same task that called `findfirst`.

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 further restrictions.

The `findnext` function is designed to be called in a loop until it fails, which indicates the last file has been found.

## Related Topics

Find **first** file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x21 service 0x50: Debugger Support

Gets address of internal program task list and sets callback vector used for program startup notification.

## Parameters

*AH*
0x50

*DX:BX*
Callback vector

*CX*
Sanity Check = 0x8765 to set callback
(Set this to zero if not a debugger calling.)

## Return Value

DX:BX contains pointer to internal task list.
AL = Task list length = 12
AH = Real-Time Interrupt rate in kHz, minus 1 kHz
(AH = 0 for 1 kHz)
CX = size of task list elements
SI = RTOS Private Data Segment
DI = Offset of RTOS idle counter
ES = 0x2 (for feature identification)

## Comments

This function was originally intended only for debugger use, but has since become useful for internal use by the Beck C-Library as well.

The callback vector in DX:BX is not accepted unless CX = 0x8765. Only debuggers should set CX to this value. All other callers should set CX to zero and then the contents of DX:BX registers passed in are not relevant.

The data structure at returned [SI:DI] pointer consists of two 32 bit unsigned long words, the first of which is the RTOS's idle loop up counter and the second is the RTOS's Real-Time Interrupt (RTI) up counter. The **RTI** rate in kHz can be calculated from the value returned in AH register by adding one.

As with all of the DOS interrupt 0x21 calls, interrupts are enabled on entry into this API.

## Beck C-Library Usage

**RTX_FineTickInit()**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x21 service 0x51: Get PSP Segment Address

Gets 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 will not work if called from tasks **created** within programs.

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 (without path name!)

## Return Value

Success: Carry flag cleared
Failure: Carry flag set, AX holds error code

AX: 1 - File not found
AX: 4 - New file name already exists
AX: 7 - Directory update failed
AX: 9 - File sharing violation
AX: 11 - Unable to sign caller into file system (too many other tasks are already using the file system)

## Comments

A file or directory cannot be renamed when it is in use by a task or program. In these cases, the failure code returned in register AX will be 7.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.16 | V1.00 |

**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
AX = File system error code

## 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

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x21 service 0x58: Get/Set memory strategy (dummy function)

Get/Set memory strategy, only a dummy function for compatibility

## Parameters

*AH*
0x58

*AL*
AL=0 Get strategy
AL=1 Set strategy.

*BX*
(Al=1) Strategy 0: First fit, 1: Best fit, 2: Last fit

## Return Value

If parameter AL == 0 (get strategy)

AX contains the memory strategy dummy value
Carry flag cleared

If parameter AL == 1 (set strategy)

IF input parameter BX >2 THEN
  AX = 1
ELSE
  AX = input parameter BX

## Comments

This is only a dummy function, added for compatibility.

The @CHIP-RTOS has its own fixed memory strategy. Memory is always allocated in the following manner:

DOS programs are always loaded into the lowest free memory block in the @CHIP-RTOS heap memory area. For memory blocks allocated internally in the @CHIP-RTOS or with **Int21h 0x48**, the @CHIP-RTOS searches for a free memory block starting at the highest heap RAM address. So the largest free memory block of the system is always located in the middle of the @CHIP-RTOS heap memory area.

The shell command **mem** shows the state of the internal memory map.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0x21 service 0x62: Get PSP Segment Address

Gets PSP (Program Segment Prefix) segment address

## Parameters

*AH*
0x62

## Return Value

BX contains the PSP segment address, if BX=0, PSP was not found

## Comments

This function will not work if called from tasks **created** within programs.

This function is identical to interrupt 0x21 service **0x51**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0x21 service 0x63: Get Leading Byte (stub)

Dummy function, not supported by the IPC@CHIP® @CHIP-RTOS

### Parameters

*AH*
0x63

### Return Value

Always returns with Carry flag set
AL = 0xFF
DS = 0
SI = 0

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

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

### Related Topics

Create New File Handle
Open Existing File

**Close** File Handle

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

End of document

---




# @CHIP-RTOS - Web Server Overview



## Web Server Overview

CGI API

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**
- **File_Upload**

## 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: Since SC12 @CHIP-RTOS V1.10 Beta, SC13 @CHIP V0.92 Beta, URL names of CGI are no longer 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`**type**data 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. **Important: The max. length of a web page in RAM produced by a users CGI function is 65519 characters.**

**Top of list**
**Index page**

---

## File Upload

It is possible to upload a file to the Web server. This feature is not available with the standard SC1x @CHIP-RTOS variants (Tiny, Small, Medium, Large, ..). If you want a special SC1x @CHIP-RTOS variant with this feature, you have to request it from us via email. At SC1x3/SC23 this feature is included at the standard RTOS and available since SC1x3 @CHIP-RTOS V1.10 Beta and SC23 @CHIP-RTOS V1.00.

The Upload will be handled using a HTML Form. The form has to include the following field:

```
DESTINATION-PATH (e.g. "A:\HELLO.EXE")
REDIRECT-PATH (e.g. "A:\WEB\SUCCESS.HTM")
FILE-CONTENT
```

The DESTINATION-PATH defines the name and the location where the file will be stored on the IPC@CHIP®'s drive. REDIRECT-PATH links to a file, which will be returned on a successful upload to the Browser. The file must be located on an IPC@CHIP®'s drive. The value of the field FILE-CONTENT will be handled from your Web Browser automatically. It allows the user to choose a local file from the PC's drive using the browse button (which appears with the type "file").

Password and user name for the File Upload are defined in the **CHIP.INI**(standard: web, web). For example your can use a form like the following one to upload a file to the IPC@CHIP®'s Web server:

```
<html>
<head>
<title>FileUpLoad to the IPC@CHIP® Web Server</title>
</head>
<body>
<form action="http://192.168.201.4" enctype="multipart/form-data" method="POST">
<p>
  Choose file from your PC for Upload:<br>
  Destination Filename:
     <input type="text" name="DESTINATION-PATH" value=""><br>
  Linkto:
     <input type="text" name="REDIRECT-PATH" value=""><br>
  Local File Path:
     <input name="FILE-CONTENT" type="file" size="50" ><br>

     <input type="submit" value=" Absenden ">
</p>
</form>
</body>
</html>

Comment:
You can also use the Type "hidden" if you want to hard code some value.
e.g: <input type="hidden" name="REDIRECT-PATH" value="success.htm">
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| on request | on request | on request | V1.10 | V1.00 |

Top of list
Index page

End of document

# @CHIP-RTOS - CGI File Types

IPC@CHIP® Documentation **Index**

## CGI File Extensions

CGI API

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, *.html | text/HTML |
| 1 | *.gif | image/gif |
| 3 | *.txt | text/plain |
| 4 | *.jpg, *.jpeg | image/jpeg |
| 5 | *.pct, *.pict | image/pict |
| 6 | *.tif, *.tiff | image/tiff |
| 10 | *.css | text/css |
| 11 | *.xml | text/xml |
| 11 | *.xsl | text/xml |
| 12 | *.wav | audio/wav |
| 13 | *.pdf | application/pdf |
| 14 | *.jar | application/java-archive |
| 16 | *.wml | text/vnd.wap.wml |
| 17 | *.wmp | image/vnd.wap.wbmp |
| 18 | *.wmc | application/vnd.wap.wmlc |
| 19 | *.wms | text/vnd.wap.wmlscript |
| 20 | *.wmx | text/vnd.wap.wmlscriptc |
| 21 | *.svg | image/svg+xml |
| 22 | *.jnl | application/x-java-jnlp-file |
| 23 | *.ico | image/x-icon |
| 24 | *.svz, *.svgz | image/svg+xml (content-encoding: gzip) |
| 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)

## Comments

The four letter file extensions are valid only for SC1x3/SC23 since SC1x3 RTOS version V1.05 and SC23 RTOS version V1.00.
On SC1x the four letter extensions can't be used.

Top of list
Index page

End of document




# @CHIP-RTOS - CGI Error Codes



## 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

# @CHIP-RTOS - CGI Examples Available

## CGI Examples

CGI

Available CGI API examples in C:

1. `minicgi.c` - Builds an HTML page
2. `countcgi.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 forms
7. `submit.c` - Building of forms

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. `cgimini.pas` - Builds an HTML page
2. `countcgi.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. **minicgi.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/minicgi
   http://192.168.205.4/minicgi?Argument
   ```

2. **countcgi.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/countcgi
```

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 input:
   ```
   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 user name and password, e.g.:
   ```
   User name: user
   Password: password
   ```

6. **submit.exe**
   The building of forms is demonstrated here.
7. **submitp.exe**
   The building of forms with method "POST" 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 @CHIP-RTOS 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 CgiMini_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 CgiMini_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 be 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. **cgiMini.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/cgimini`
   `http://192.168.205.4/cgimini?Argument`

2. **countcgi.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: `http://192.168.205.4/countcgi`

---

End of document

---

# @CHIP-RTOS - Data Structures used in CGI API

## 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.

**Contents** :


- typedef CGI_Entry
- define CGI HTTP Request
- typedef FormItem
- typedef rpCgi
- typedef CUSTOM_MIMETYPES_TABLE
- typedef CUSTOM_CONTENTENCODING_TABLE


---

## typedef 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 enumeratorused 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**

---

## define 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**

---

## typedef FormItem

```
typedef struct tag_form_item
{
    char   *NamePtr;
    char   *ValuePtr;
} FormItem;
```

### Comments

Both strings referenced here are null terminated.

### Related Topics

API function **CGI_GETFORMITEM** - Split a form item into name and value

# typedef rpCgi

```
typedef struct {
    //*****************************************************
    //    Request fields  (Read Only!!!!)
    //*****************************************************
    unsigned char  fConnectionId;          // SC1x3/SC23: 0=HTTP/1=HTTPS (SSL) server
                                           // SC1x: -- 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;           // points to the remoteIP,
                                           // you must split the octets
                                           // For using the IP to etsablish TCP/IP
                                           // connections, you have to exchange
                                           // lowbyte and highbyte!

    //*****************************************************
    //    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
                                           // (on SC1x: max. length is 65519 chars)
    unsigned long  fObjectDate;            // -- internal, do not modify --
    unsigned int   fHostIndex;             // -- internal, do not modify --

} rpCgi, *rpCgiPtr;
```

## Comments

**`fHttpResponse`**

This is an enumeration type that specifies the web server response.
You should use one of the macros defined in the CLIB:

```
#define  CgiHttpOk               0  //  CGI returns HTTP 200 Ok
#define  CgiHttpOkStatic         1  //  CGI returns HTTP 200 Ok - Static Object
#define  CgiHttpRedirect         2  //  CGI returns HTTP 302 Moved Temp
#define  CgiHttpNotModified      3  //  CGI returns HTTP 304 Not Modified
#define  CgiHttpUnauthorized     4  //  CGI returns HTTP 401 Unauthorized
#define  CgiHttpNotFound         5  //  CGI returns HTTP 404 Not Found
#define  CgiHttpOKNoDoc          6  //  CGI returns HTTP 204 No document follows
#define  CgiHttpNotAvail         7  //  CGI returns HTTP 503 Service Not Available
#define  CgiHttpSrvError         8  //  CGI returns HTTP 500 Internal Srv Error
```

**`fDataType`**

This is an enumeration type that specifies the filetype.
You should use one of the macros defined in the CLIB:

```
#define  CGIDataTypeHtml                0  // CGI returns  text/html
#define  CGIDataTypeImageGif            1  // image/gif
#define  CGIDataTypeApplet              2  // application/octet-stream
#define  CGIDataTypeText                3  // text/plain
#define  CGIDataTypeImageJpeg           4  // image/jpeg
```

```
#define  CGIDataTypeImagePict             5  // image/pict
#define  CGIDataTypeImageTiff             6  // image/tiff
#define  CGIDataTypeImagePng              7  // image/png
#define  CGIDataTypeForm                  8  // application/x-www-form-urlencoded
#define  CGIDataTypeIpp                   9  // application/ipp
#define  CGIDataTypeCss                  10  // text/css
#define  CGIDataTypeXml                  11  // text/xml
#define  CGIDataTypeWav                  12  // audio/wav
#define  CGIDataTypePdf                  13  // application/pdf
#define  CGIDataTypeJavaArchive          14  // application/java-archive
#define  CGIDataTypeOctet                15  // application/octet-stream
#define  CGIDataTypeVndWapWml            16  // text/vnd.wap.wml
#define  CGIDataTypeVndWapWbmp           17  // image/vnd.wap.wbmp
#define  CGIDataTypeVndWapWmlc           18  // application/vnd.wap.wmlc
#define  CGIDataTypeVndWapWmlscript      19  // text/vnd.wap.wmlscript
#define  CGIDataTypeVndWapWmlscriptc     20  // text/vnd.wap.wmlscriptc
#define  CGIDataTypeScaleVectGraph       21  // image/svg+xml
#define  CGIDataTypeScaleVectGraphZipped 24  // image/svg+xml (content encoding: gzip)
```

**fRemoteIPPtr**

This is an unsigned long value that represents the IPv4 address of the remote host. To get the string representation of the IP address use the following code:

**On SC1x:**

```
sprintf( ipString, "%d.%d.%d.%d",
     (int)( ( *( CgiRequest->fRemoteIPPtr ) & 0xFF000000l ) >> 24 ),
     (int)( ( *( CgiRequest->fRemoteIPPtr ) & 0x00FF0000l ) >> 16 ),
     (int)( ( *( CgiRequest->fRemoteIPPtr ) & 0x0000FF00l ) >> 8 ),
     (int)( *( CgiRequest->fRemoteIPPtr ) & 0x000000FFl ) );
```

**On SC1x3/SC23:**

```
InetToAscii((const unsigned long far *)CgiRequest->fRemoteIPPtr, ipString);
```

## Related Topics

CGI Callback **Function**

**Top of list**
**Index page**

---

# typedef CUSTOM_MIMETYPES_TABLE

```
typedef struct
{
  unsigned int mimeId;    // enum value, >= 100 for new types
  char far*    mimeExt;   // file extension, e.g. "gif"
  char far*    mimeType;  // mime type, e.g. "image/gif"
} CUSTOM_MIMETYPES_TABLE;
```

## Comments

**mimeId**

This is an enumeration type. If the value is equal to the **internal used fDatatype value**, the MIME type value is overwritten for this ID. For new custom MIME types, values greater 100 should be used.

**mimeExt**

On Sc1x only three letter extensions are allowed.

## Related Topics

API function **CGI_MIME_INSTALL** - Install custom MIME table

# typedef CUSTOM_CONTENTENCODING_TABLE

```
typedef char far *CUSTOM_CONTENTENCODING_TABLE;
```

## Comments

Contains the null terminated string which will be send to the browser. A Null pointer or an empty string will be identified as no encoding type. In that case the http content-encoding header will not created.

## Related Topics

API function **CGI_CONTENT_ENCODING_INSTALL** - Install custom content encoding table



End of document




# @CHIP-RTOS - Data Structures used in BIOS API



## Data Structures

Here are the data structures used by the @Chip-RTOS BIOS **API**. All constants and data structures are defined in the header file BIOS_API.H

Notes:

- **Byte alignment is required** for all data structures used within the API.

**Content** :


- **BiosDiskInfo**
- **FILE_FIND**
- **FS_FILETIME**
- **LFN_FILE_FIND**


## BiosDiskInfo

```
typedef struct
{
    unsigned int  bytesPerSector;
    unsigned int  sectorsPerCluster;
    unsigned long freeClusters;
    unsigned long totalClusters;
} BiosDiskInfo;
```

### Comments

This structure is defined in the header file BIOS_API.H. The BIOS SWI 0xA0 **service 0x0A** fills in this structure's data.

The number of bytes disk free space can be calculated as

```
FreeClusters * sectorsPerCluster * bytesPerSector
```

and the drive's total number of bytes as

```
TotalClusters * sectorsPerCluster * bytesPerSector
```

Note that the sector size used by the file system is fixed at 512 bytes.

## Related Topics

API function **BIOS_DISK_INFO** Get Disk Info

**Top of list**
**Index page**

---

# FILE_FIND

```
typedef struct filefind {

    char           filename[12];   // Null terminated filename
    char           fileext[4];     // and extension
    unsigned short fileattr;       // MS-DOS file attributes
    short int      reserved;       // Reserved

    FS_FILETIME filetimestamp;     // Time & date last modified

    unsigned long   filesize ;     // File size
    char    szShort_Name[FS_SHORTBUF] ;

    // private_field[] area is used internally by the @Chip-RTOS.
    char            private_field[FFIND_PRIVATE_SIZE];

} FILE_FIND ;
```

## Comments

This structure is defined in the header file BIOS_API.H. From the user's point of view this structure is read-only data. No fields need to be initialized by the user prior to the **BIOS_Fast_Findfirst** call. The @Chip-RTOS does the writing to this structure.

**`filename`**
A null terminated ASCII string containing the base portion of the file name prior to any extension part. Note: This string length is at most 8 characters plus the null terminator in spite of the 12 byte declaration.

**`fileext`**
A null terminated ASCII string containing the extension portion of the file name. This string may be empty if the file name has no extension.

**`fileattr`**
A bit field that specifies the file's attributes as defined by the following constants declared in BIOS_API.H:

```
#define FJ_DA_NORMAL     0x00
#define FJ_DA_RDONLY     0x01
#define FJ_DA_HIDDEN     0x02
#define FJ_DA_SYSTEM     0x04
#define FJ_DA_VOLUME     0x08
#define FJ_DA_DIRENT     0x10
#define FJ_DA_ARCHIVE    0x20
```

**reserved**
This structure member is not used.

**filetimestamp**
The **structure**contains the time and date at which this file was last modified.

**filesize**
The total number of bytes in the file.

**szShort_Name**
This 14 byte buffer will hold a null terminated ASCII string containing the DOS file name. This string can be up to 12 characters in length plus a terminating null character. (Same information as in `filename`and `fileext`presented in a single string.)

**Note:** This field is not present in SC1x @Chip-RTOS versions prior to SC1x version 1.20 @Chip-RTOS.

**private_field**
Internal state information needed by the @Chip-RTOS is held here. The user must not change anything in this area between the **BIOS_Fast_Findfirst**call and the last **BIOS_Fast_Findnext**or **BIOS_Fast_Finddone**call.

**Note:** The size of this private field has been substantially reduced as of SC1x version 1.20 @Chip-RTOS. When compatibility with older @Chip-RTOS versions is required by the user for some reason, then the `FILEFIND_OLD_RTOS`compilation constant must be defined prior to including the BIOS_API.H include file to restore this interface data structure back to its original size. For example:

```
#define FILEFIND_OLD_RTOS
#include "clib.h"
```

## Related Topics

API function **BIOS_FAST_FINDFIRST** Fast findfirst
API function **BIOS_FAST_FINDNEXT** Fast findnext
API function **BIOS_FAST_FINDDONE** Fast finddone
**LFN_FILE_FIND** - Long File Name find structure

**Top of list**
**Index page**

---

# FS_FILETIME

```
typedef struct tag_filetimestamp
{
    unsigned short int filedate;
    unsigned short int filetime;
} FS_FILETIME ;
```

## Comments

This structure is defined in the header file BIOS_API.H.

**filedate**

This is a bit field which encodes the files date as follows:

B0-B4: Day of month in range 1..31
B5-B8: Month of year in range 1..12
B9-B15: Year - 1980

or in other words:

filedate = ((year - 1980) shl 9) or (month shl 5) or day ;

**filetime**

This is a bit field which encodes the files date as follows:

B0-B4: Seconds/2 in range 0..29 (seconds scaled @B-1)
B5-B10: Minute in range 0..59
B11-B15: Hour in range 0..23

or in other words:

filetime = (hour shl 11) or (min shl 5) or (sec / 2);

## Related Topics

API function **BIOS_FAST_FINDFIRST** Fast findfirst
API function **BIOS_LFN_FINDFIRST** Long file name findfirst

**Top of list**
**Index page**

---

# LFN_FILE_FIND

```
typedef struct LFN_filefind {
    char           *pszLong_Name ;          // FS_MAXLONGPATH size

    char            filename[12];           // Null terminated filename
    char            fileext[4];             // and extension
    unsigned short  fileattr;               // MS-DOS file attributes
    short int       reserved;               // Reserved

    FS_FILETIME filetimestamp;              // Time & date last modified

    unsigned long   filesize ;              // File size
    char            szShort_Name[FS_SHORTBUF] ; // DOS Alias

    // private_field[] area is used internally by the @Chip-RTOS.
    char            private_field[40];

} LFN_FILE_FIND ;
```

## Comments

This structure is defined in the header file BIOS_API.H. From the user's point of view this structure is read-only data. No fields need to be initialized by the user prior to the **BIOS_LFN_Findfirst** call. The @Chip-RTOS does the writing to this structure.

**pszLong_Name**

The null terminated ASCII file name string of up to 260 characters is reported here by the long file

name findfirst/findnext functions. This memory is allocated by the @Chip-RTOS API and not by the caller. Note that a final call to the **BIOS_LFN_Find_Done**function is required in order to release this memory unless either the **BIOS_LFN_Findfirst**or **BIOS_LFN_Findnext**had returned FALSE in which case the memory has already been released.

**filename**
A null terminated ASCII string containing the base portion of the short DOS 8.3 alias file name prior to any extension part. Note: This string length is at most 8 characters plus the null terminator in spite of the 12 byte declaration.

**fileext**
A null terminated ASCII string containing the extension portion of the short DOS 8.3 alias file name. This string may be empty if the alias has no extension.

**fileattr**
A bit field that specifies the file's attributes as defined by the following constants declared in BIOS_API.H:

```
#define FJ_DA_NORMAL     0x00
#define FJ_DA_RDONLY     0x01
#define FJ_DA_HIDDEN     0x02
#define FJ_DA_SYSTEM     0x04
#define FJ_DA_VOLUME     0x08
#define FJ_DA_DIRENT     0x10
#define FJ_DA_ARCHIVE    0x20
```

**reserved**
This structure member is not used.

**filetimestamp**
The **structure**contains the time and date at which this file was last modified.

**filesize**
The total number of bytes in the file.

**szShort_Name**
This 14 byte buffer will hold a null terminated ASCII string containing the DOS 8.3 alias file name. This string can be up to 12 characters in length plus a terminating null character. (Same information as in `filename`and `fileext`presented in a single string.)

**private_field**
Internal state information needed by the @Chip-RTOS is held here. The user must not change anything in this area between the **BIOS_LFN_Findfirst**call and the last **BIOS_LFN_Findnext**or **BIOS_LFN_Find_Done**call.

## Related Topics

API function **BIOS_LFN_FINDFIRST** Long file name findfirst
API function **BIOS_LFN_FINDNEXT** Long file name findnext
API function **BIOS_LFN_FINDDONE** Long file name finddone

**Top of list**
**Index page**

End of document

## BIOS_Disk_Info

Get disk drive information.

```
void BIOS_Disk_Info ( unsigned int drive,
                      BiosDiskInfo * disk_info );
```

## Parameters

*drive*

Drive index:
0 = current drive
1 = A: drive
2 = B: drive
... and so on.

*disk_info*

Output parameter: Pointer to **BiosDiskInfo** data structure that will be set by this API.

## Return Value

-- none --

## Comments

This function may be used for any drive, but it will be required for drives which exceed the DOS interrupt 0x21 **service 0x36** API's reporting capacity of 0xFFFF clusters maximum.

The resulting value for the `SectorsPerCluster` member of the output data structure will be 0xFFFF if the drive specified was not valid.

The number of bytes disk free space can be calculated as

```
FreeClusters * sectorsPerCluster * bytesPerSector
```

and the drive's total number of bytes as

```
TotalClusters * sectorsPerCluster * bytesPerSector
```

## See Also

BIOS_Disk_Space

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.16 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.11 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---

## BIOS_Disk_Space

Report number of free bytes on specified disk drive.

```
unsigned long BIOS_Disk_Space (unsigned int drive );
```

## Parameters

*drive*

Drive index:
0 = current drive
1 = A: drive
2 = B: drive
... and so on.

## Return Value

Available disk space, number of bytes free. This value will be zero if the drive number specified was invalid.

## Comments

This function may be used for any drive, but it will be required for drives which exceed the DOS interrupt 0x21 **service 0x36** API's reporting capacity of 0xFFFF clusters maximum.

## See Also

**BIOS_Get_Disk_Info**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.16 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.11 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Set_Ini_String

Insert an entry into CHIP.INI.

```
int BIOS_Set_Ini_String ( const char far *section,
                          const char far *item_name,
                          const char far *item_text );
```

## Parameters

*section*

Pointer to section string (max. 40 chars)

*item_name*

Pointer to item name (max. 40 chars)

*item_text*

Pointer to item text (max. 128 chars)
Pass a NULL pointer here to erase the complete item line

## Return Value

0: success
-1: Invalid string length, low heap memory or file error.

## Comments

Leading and trailing white space characters will be cut. The function stops reading the entry when it finds an invalid character (<0x20).
This API is thread safe and reentrant. This function will fail (no operation) if insufficient system heap memory is available to momentarily hold the contents of the CHIP.INI file.

Example usage:

```
BIOS_Set_Ini_String ("MY_SECTION", "MY_ITEM", "VALUE_TEXT") ;

... produces the following CHIP.INI entry

[MY_SECTION]
MY_ITEM=VALUE_TEXT
```

## See Also

BIOS_Get_Ini_String

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## BIOS_Get_Ini_String

Get an entry from CHIP.INI.

```
int BIOS_Get_Ini_String ( const char far *section,
                          const char far *item_name,
                          char far *item_text,
                          unsigned int max_len );
```

## Parameters

*section*

Pointer to section string (max. 40 chars)

*item_name*

Pointer to item name (max. 40 chars)

*item_text*

Output parameter: Pointer to where up to `max_len` characters of item text will be written, plus a terminating nil character.

*max_len*

Maximum characters to output to `item_text` , not including terminating nil character.

**Caution:** Buffer at `item_text` must be `max_len` + 1 size.

## Return Value

0: Entry not found
>0 : Length of the string found
-1 : CHIP.INI file not found

## Comments

This API is thread safe and reentrant. This function will fail (returns -1) if insufficient system heap memory is available to momentarily hold the contents of the CHIP.INI file.

Example usage:

```
// Declare an output buffer to receive the item's text
unsigned char item_text[100] ;

unsigned int text_length =
BIOS_Get_Ini_String("MY_SECTION", "MY_ITEM",item_text, sizeof(item_text)-1);
```

## See Also

**BIOS_Set_Ini_String**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---



# @CHIP-RTOS - Data Structures used in TCP/IP API



## Data Structures

Here are the BSD structures and other data types used by the **TCP/IP API**.
All constants and data structures are defined in the TCPIPAPI.H header file.

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).

**Contents** :


- ArpHeader
- ArpRouteCacheEntry
- ArpUserCallbackInfo
- atEntry
- Crypt_MD5_Context
- Crypt_SHA1_Context
- Crypt_GenericKey
- Crypt_AES_Context
- DevUserBuffer
- DevUserBufferHandle
- DevUserDriver
- DevUserIfaceHandle
- DevIpv4IfaceCfg
- DevIpv6IfaceCfg
- DevUserLinkLayer
- DevPPPOption
- DnsMailHost
- fd_set
- GetSocketOption
- IcmpMib
- Iface_Entry
- Iface_Entry_Ext
- IfaceIPv6_Entry
- IfMib
- in_addr
- in6_addr
- IpMib
- IpUserCallbackInfo
- Packet_Count
- PasCallBack
- Ping
- PKI_CERT_ADD_Entry
- PKI_OWNKEYPAIR_ADD_Entry
- PPPClie_ModemCmd
- PPPClie_ModemCtrl
- PPPClient_Init
- PPPClient_Init_IPv6
- PPPDial
- PPP_IPCfg_Data

- PPP_IPCfg_Data_IPv6
- PPP_ModemHangup
- PPP_Option
- recv_params
- Route_Entry
- send_params
- SetSocketOption
- sockaddr
- sockaddr_in
- sockaddr_in6
- sockaddr_storage
- SocketInfo
- SocketInfoIpv4v6
- SSL_SESSION_NEW_Entry
- TcpMib
- timeval
- UdpMib
- UserEthDhcp_Entry

---

# ArpHeader

*Address Resolution Protocol* (ARP) packet header has following form.

```
typedef struct tag_ArpHeader
{
    unsigned int     arpHardware;
    unsigned int     arpProtocol;
    unsigned char    arpHardwareLength;
    unsigned char    arpProtocolLength;
    unsigned int     arpOperation;
    unsigned char    arpSendPhyAddress[6];
    unsigned char    arpSendNetAddress[4];
    unsigned char    arpTargetPhyAddress[6];
    unsigned char    arpTargetNetAddress[4];
} ArpHeader;
```

## Related Topics

API function ARP_USER_CB - Install ARP callback function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# ArpRouteCacheEntry

```
typedef struct tagArpRouteCacheEntry
{
    unsigned int   ifIndex;       // Interface index for this entry
    unsigned long  flags;         // Type/state of entry (see below)
    unsigned long  arpIpAddress;  // Device IP address or network address
    unsigned long  mask;          // Ip network mask for cloned ARP entries
    long           arpTtl;        // TimeToLive in milliseconds
    long           hops;          // Number of routers between this host and route

    union
    {
        struct
        {
            // Device IP address for Local routing entry
            unsigned long  DevIpAddress;
        } LocalNetRouteEntry;
        struct
        {
            // Device Ethernet address for ARP entry
            unsigned char MacAddress[6];
            unsigned char filler[2];
        } ArpEntry;
        struct
        {
            unsigned long  GatewayIpAddress;   // Indirect route
        } GatewayEntry;
    } ArpRouteGwayUnion;
} ArpRouteCacheEntry;
```

## Comments

**`flags`**

This bit field is defined as follows:

Note: Bit 2, 3, and 4 select the view of the `ArpRouteGwayUnion`union.
Bit 2: Indirect route (`GatewayEntry`structure)
Bit 3: Valid ARP entry (`ArpEntry`structure)
Bit 4: Local route (`LocalNetRouteEntry`structure)
Bit 5: Static route
Bit 10, 11: Cloned ARP entry (e.g. ARP entry for default gateway)
Bit 16: Route is up
Bit 17: `arpIpAddress`entry is a host address, 0: network address

## Related Topics

API function **GET_ARPROUTE_CACHE** - Read internal ARP/Route cache table

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# ArpUserCallbackInfo

A structure of this type is passed as an input parameter to an ARP user callback handler.

```
typedef struct ArpUserCallbackInfo
{
   int              size;         // Size of this structure
   unsigned int     dataLength;   // Length of the data in the ARP package
   void    far *    dataPtr;      // Pointer to the ARP data
   void    far *    devHandlePtr; // Device handle pointer of this incoming packet
                                  // for detecting the source device
} ArpCallbackUserInfo_t;
```

### Related Topics

ArpHeader data structure
API function ARP_USER_CB - Install ARP callback function

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

Top of list
Index page

---

## atEntry

```
typedef struct tagAtEntry {
    long           IfIndex;         // Interface on which this entry maps
    unsigned char  PhysAddress[11]; // Physical address of destination
    unsigned char  PhysAddressLen;  // Length of PhysAddress
    unsigned long  NetAddress;      // IP address of physical address
} atEntry;
```

### SC1x Comments

These structures are only available in SC1x @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to the internal TCP/IP SNMP variables.

### Related Topics

API function API_SNMP_GET - Get internal TCP/IP SNMP variables

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Crypt_MD5_Context

```
typedef struct tag_crypt_md5_context
{
  unsigned long i[2];
  unsigned long buf[4];
  unsigned char in[64];
  unsigned char digest[16]; // actual digest after MD5Final call
} Crypt_MD5_Context;
typedef Crypt_MD5_Context far * Crypt_MD5_ContextPtr;
```

### Related Topics

API function **CRYPT_HASH** Build MD5/SHA1 checksum

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## Crypt_SHA1_Context

```
typedef struct tag_crypt_sha1_context
{
  unsigned long shaDigest[5];
  unsigned long shaCountLo;
  unsigned long shaCountHi;
  unsigned long shaData[16];
  unsigned char digest[20];  // actual digest after SHA1Final call
} Crypt_SHA1_Context;
typedef Crypt_SHA1_Context far * Crypt_SHA1_ContextPtr;
```

### Related Topics

API function **CRYPT_HASH** Build MD5/SHA1 checksum

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

**Top of list**
**Index page**

---

## Crypt_GenericKey

```
typedef struct tag_crypt_GenericKey
{
    unsigned char far *  keyDataPtr;
    int                  keyRounds;  // not used for AES
    unsigned int         keyBits;
} Crypt_GenericKey;
typedef Crypt_GenericKey far * Crypt_GenericKeyPtr;
```

### Related Topics

API function **CRYPT_CIPHER** Use cipher functions

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

Top of list
Index page

---

## Crypt_AES_Context

```
typedef struct tag_crypt_aes_context
{
    unsigned long far rd_key[60];
    int rounds;
} Crypt_AES_Context;
typedef Crypt_AES_Context far * Crypt_AES_ContextPtr;
```

### Related Topics

API function **CRYPT_CIPHER** Use cipher functions

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

Top of list
Index page

---

## DevUserBuffer

This buffer handle private type is used internally by the TCP/IP stack.

```
typedef void  far *  DevUserBuffer;          // Device buffer handle type
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## DevUserBufferHandle

This type references a buffer handle private type used internally by the TCP/IP stack.

```
typedef DevUserBuffer far *  DevUserBufferHandle; // Pointer to a device buffer handle
```

### Related Topics

**DevUserBuffer** type - Device buffer handle (private)
API function **DEV_GET_BUF** - Get a buffer from TCP/IP stack

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## DevUserDriver

```
typedef struct tag_DevUserDriver
{
    int                     DevIndex;        // Internal device index, filled by @CHIP-RTOS
    char far *              DevName;         // Unique device name, max. 13 chars  + 0
    unsigned long           IpAddr;          // IP address
    unsigned long           Netmask;         // Netmask

    DevUserIfaceHandle      IfaceHandle;     // handle to identify the interface internal

    DevUserLinkLayer        LinkLayerHandle; // handle to identify the link layer internal


    unsigned char           iface_type,      // Type of the device driver:
        // 0: unknown user specific device,
        // 1 = DEV_ETHERNET: device is an Ethernet controller,
        // 2 = DEV_PPP_SERVER: use PPP protocol as a server
        // 3 = DEV_PPP_CLIENT: use PPP protocol as a client
        // 4: use SLIP protocol (currently not supported)
        // 5 = DEV_PPPoE_CLIENT: use PPP protocol over Ethernet as a client  (SC1x3/SC23 only)
        //   Note: The DEV_ADD_ONLY flag may be OR'ed into iface_type on first
        //   call to DEV_OPEN_IFACE API to postpone the device configuration
        //   until a subsequent call.

    unsigned char           use_dhcp;        // Boolean, set to 1 if you want to use
                                             //  DHCP server IP configuration.

    UserEthDhcp_Entry far * Dhcp_Data;       // Pointer to DHCP data set by
                                             //  DEV_WAIT_DHCP_COMPLETE API
                                             //  (only applies if use_dhcp=1
                                             //   and iface_type=1 =Ethernet).


    void far *              PPPLinkNotify;

    PPP_Option  far *       PPPoptions;

    void  far *             Reserved3;

    // Set of driver functions which user implements:

    void  far *             DevOpen;         // Driver Open/initialize function,
                                             //  optional (pass a Null pointer if not used).
    void  far *             DevClose;        // Driver Close function,
                                             //  optional (pass a Null pointer if not used).
    void  far *             DevSend;         // Driver send function
    void  far *             DevRecv;         // Driver recv function
    void  far *             DevGetPhysAddr;  // Ethernet driver only, returns MAC address,
                                             //  else pass a Null pointer.
    void  far *             DevFreeRecv;     // Free own allocated buffer,
                                             //  optional (pass a Null pointer if not used).
    void  far *             DevIoctl;        // Driver specific function, e.g. informs about ip config update
```

```
    int   far *            ErrorCode;      // Contains error code, if driver install
                                           //  (API call 0xAC 0xA0) failed.

    void  far *            Reserved4;      // reserved for future extensions
    void  far *            Reserved5;
    void  far *            Reserved6;

} DevUserDriver;
```

## Comments

The `DevUserDriver` is the main structure which is needed for installing your own device drivers/interfaces for the TCP/IP stack of the @CHIP-RTOS. Further explanations for certain structure members follows.

**`PPPLinkNotify`**

This member is ignored unless the `iface_type`is set to 2 or 3 (PPP server or client). This is a pointer to a user provided "link notify" callback function, which is called by the PPP layer to inform the user about the current PPP device status. Set this pointer to NULL if a link notify callback from the the PPP layer is not required by your driver.

The callback function should have the form:

```
void huge PPPnotify(DevUserIfaceHandle ifaceHandle, int flags);
```

The function is called with the interface handle and a set of flags. The flags consists of one or more of the following bits OR'ed together:

0x0002: PPP device is ready to accept data from the user
0x0008: Close procedure started
0x0010: PPP device has closed
0x0020: LCP layer is up
0x0040: PAP is up
0x0080: CHAP is up

**`PPPoptions`**

Pointer to the first PPP option (struct **PPP_Option**) of a user provided PPP option array. These options will be set inside the **DEV_OPEN_IFACE**call. If the setting of the option failed, the `Dev_Open_Interface`exits with error 222 or 242.

As an end marker, the element `protocol_level`of the last option of the array must be 0.

Set this pointer to null, if you have no PPP options.

## Related Topics

API function **DEV_OPEN_IFACE** - Open/install a user TCP/IP device driver/interface
**DevUserIfaceHandle** type - `IfaceHandle` structure member
**DevUserLinkLayer** type - `LinkLayerHandle` structure member
**UserEthDhcp_Entry** type - `Dhcp_Data` structure member

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# DevUserIfaceHandle

A handle of this type is returned in the `IfaceHandle` member of the `DevUserDriver` **structure** used in the DEV_OPEN_IFACE API.

```
typedef void far *  DevUserIfaceHandle;   // handle type, to identify a user interface
```

### Comments

The data referenced by this handle is private and internal to the @CHIP-RTOS TCP/IP stack.

### Related Topics

API function **DEV_OPEN_IFACE** - Open/install a user TCP/IP device driver/interface

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## DevIpv4IfaceCfg

```
typedef struct tag_DevIpv4IfaceCfg
{
  DevUserIfaceHandle     IfaceHandle;     // handle to identify the interface internal
  unsigned long          IpAddr;          //IP address
  unsigned long          Netmask;         //Netmask
  int                    flags;
  int                    reserved;
  unsigned char          multiHomeIndex;   //The particular index number for this IP address (multihoming)
}DevIpv4IfaceCfg;
```

### SC1x Comments

**flags**
This bit field is defined as follows:

Bit 7: IP forwarding enabled
Bit 8: IP forwarding of directed broadcast enabled
Bit 11: Configure IP address via DHCP protocol

### SC1x3/SC23 Comments

**flags**
This bit field is defined as follows:

Bit 7: Configure IP address via DHCP protocol
Bit 10: IP forwarding enabled
Bit 11: IP forwarding of directed broadcast enabled

### Related Topics

API function **DEV_CONFIG_IFACE** -

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## DevIpv6IfaceCfg

```
typedef struct tag_DevIpv6IfaceCfg
{
  DevUserIfaceHandle               IfaceHandle;    // handle to identify the interface internal
  const struct sockaddr_storage *  IpAddrPtr;      //IP address
  int                              prefixlen;      // Prefix Length
  int                              reserved;
  int                              ipv6flags;
  void far *                       reservedPtr;     //reserved, currently not used
  unsigned char                    multiHomeIndex;  //The index for this IP address for multihoming
}DevIpv6IfaceCfg;
```

### Comments

**`ipv6flags`**
This bit field is defined as follows:
Bit 11:  Enables auto interface ID

### Related Topics

API function DEV_CONFIG_IFACE_IPv6 -

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | 1.05 | V1.00 |

Top of list
Index page

---

## DevUserLinkLayer

```
typedef void  far *  DevUserLinkLayer;     // handle type to identify link layer,
                                           //  (needed for internal use)
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# DevPPPOption

This data structure and specified constants are defined in the TCPIPAPI.H library header file.

```
typedef struct tag_devpppopt
{
    DevUserIfaceHandle     IfaceHandle;     // handle to identify the interface internal
    int                    protocol_level;
    int                    remoteLocalFlag;
    int                    optionName;       // Option's name
    const  char   far     *optionValue;      // the value of the option
    int                    optionLength;     // Length of the option
}DevPPPOption;
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# DnsMailHost

Holds information on a mail host.

```
typedef struct DnsMailHost_
{
  char name[DNS_SIZE_NAME_STRING + 1];  // Domain name of the host
  unsigned short preference;            // Preference value of the host. Lower values are preferred.
} DnsMailHost;
```

### Related Topics

API function DNS_GET_MAIL_HOSTS

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

Top of list
Index page

---

# fd_set

```
typedef unsigned int fd_mask ;

typedef struct fd_set
{
    fd_mask fds_bits[MAX_BECK_SOCKETS/16] ;    // Socket bit field
} fd_set;
```

### Comments

This data structure provides a bit field to specify which sockets to monitor and which sockets events occur in the **select()** API. Each bit corresponds to a socket. The socket descriptor is used to provide the mapping between bits and sockets. For example for a socket with descriptor zero, this socket will be represented by bit 0 in this field.

A set of macros are provided in the Beck C-library TCPAPI.H header file for manipulating these bit fields:

**FD_SET(n, p)** - Sets bit 'n' in `fd_set` at pointer 'p' to 1.
**FD_CLR(n, p)** - Zeroes bit 'n' in `fd_set` at pointer 'p'.
**FD_ISSET(n, p)** - Tests bit 'n' in `fd_set` at pointer 'p'.
**FD_ZERO(p)** - Clears all bits in `fd_set` at pointer 'p'.
**FD_COPY(src, dest)** - Copies object at pointer 'src' into the object at pointer 'dest'. The intent here is that both of these pointers reference a `fd_set` structure, but the compiler will not type check this for you.

Also the socket iterator function **FD_Next()** is available to sequence through a set of sockets.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

**Top of list**
**Index page**

---

## GetSocketOption

This data structure and specified constants are defined in the TCPIPAPI.H library header file.

```
typedef struct tag_getsockopt
{
    int         protocol_level;   // Protocol level: IP level, TCP level or socket level
    int         optionName;       // Option's name
    char  far  *optionValue;      // Pointer to the option value buffer (type varies)
    int   far  *optionLength;     // Length of option value buffer

} GetSocketOption;
```

### Comments

The type of data found at the location referenced by the `optionValue` member pointer 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 (in bytes) pointed to by the `optionValue` member depends on the option, and this size must be specified in the `optionLength` field. These sizes are stated in the list of options below. The API uses the `optionValue` and `optionLength` members to report option values.

The following list specifies 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 header file. **Important** : The constants for IPC@CHIP® SC1x and SC1x3/SC23 are different. Take a look at the accordant target sections at tcpipapi.h. Use always the defined constant names.

**`protocol_level=IP_PROTOIP_LEVEL`** (IP level options)

- optionName=`IPO_TOS`: Size: 8 bit, sizeof(char)
  Set IP type of service.
  default: 0

- optionName=`IPO_TTL`: Size: 8 bit, sizeof(char)
  Set IP time-to-live router hops.
  default: 64

**`protocol_level=IP_PROTOTCP_LEVEL`** (TCP level options)

- optionName=TCP_NODELAY: Size: 16 bit, sizeof(int)
  1: Disable nagle algorithm / 0: Enable nagle algorithm
  default: 0

- optionName=TCP_NOPUSH: Size: 16 bit, sizeof(int)
  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 as with FTP).
  default: 0

- optionName=TCP_SLOW_START: Size: 16 bit, sizeof(int)
  1: Enable the TCP slow start algorithm / 0: disable
  default: 1

- optionName=TCP_KEEPALIVE: Size: 16 bit, sizeof(int)
  Set idle time seconds before sending keep alive probes.
  default: 7200, minimum: 10, maximum: 32767 seconds
  *Notes:*
  1. The socket level option SO_KEEPALIVE (see below) must be enabled.
  2. The minimum value was changed in @CHIP-RTOS V0.70 to 10 seconds (was 7200 seconds in @CHIP-RTOS V0.69).

- optionName=TCP_DELAY_ACK: Size: 16 bit, sizeof(int)
  Set the TCP delay ACK time in milliseconds.
  default: 200, minimum: 0, maximum: 65535 ms

- optionName=TCP_FINWT2TIME: Size: 16 bit, sizeof(int)
  Set the maximum amount of time TCP will wait for the remote side to close.
  default: 600, minimum: 0, maximum: 32767 seconds

- optionName=TCP_2MSLTIME: Size: 16 bit, sizeof(int)
  Set the maximum amount of time TCP will wait in the TIME WAIT state, once it has initiated a close.
  default: 2, minimum 0, maximum 32767 seconds

- optionName=TCP_MAXRT: Size: 16 bit, sizeof(int)
  Set the TCP/IP timeout in seconds, (0=System Default [75 Seconds], -1 = endless).
  default: 75, minimum: 1, maximum: 32767 seconds or endless.

- optionName=TCP_MAXREXMIT: Size: 16 bit, sizeof(int)
  Set the maximum number of TCP/IP send retries.
  default 12, minimum 0, maximum 32767

- optionName=TCP_KEEPALIVE_INTV: Size: 16 bit, sizeof(int)
  Set keep alive interval probes.
  default: 75, minimum: 1, maximum: 600 seconds
  *Notes:*
  1. This value can not be changed after a connection is established.
  2. Also the socket must not be in listen mode.

- optionName=TCP_KEEPALIVE_CNT: Size: 16 bit, sizeof(int)
  Set maximum number of keep alive probes before TCP gives up and closes the connection.
  default: 8, minimum: 0, maximum: 32767

- optionName=TCP_SSL_CLIENT: Size: 16 bit, sizeof(int)
  Enable SSL client negotiation, 1: enable, 0: disable
  default: 0

- optionName=TCP_SSL_SERVER: Size: 16 bit, sizeof(int)
  Enable SSL server negotiation, if set for a listening socket, all accepted sockets inherit this option value, 1: enable, 0: disable
  default: 0

- optionName=TCP_SSLSESSION: Size: 16 bit, sizeof(int)
  Set session Id for socket, if set for a listening socket, all accepted sockets inherit this option value

- optionName=TCP_SSL_SEND_MIN_SIZE: Size: 16 bit, sizeof(int)
  Set the SSL send minimum size, if users send data is less than this value, user data will be queued.
  default: 0 (no queue), minimum 0, maximum 65535

- optionName=TCP_SSL_SEND_MAX_SIZE: Size: 16 bit, sizeof(int)
  Set the SSL record maximum size, each record will at most have that much user data encapsulated.
  User data bigger than that size limit will be cut into two records.
  default: 8000 (no queue), minimum 100, maximum 16384

- optionName=TCP_RTO_DEF: Size: 32 bit, sizeof(long)
  Set the default retransmission timeout value in milliseconds.
  default: 3000 ms.

- optionName=TCP_RTO_MIN: Size: 32 bit, sizeof(long)
  Set the minimum retransmission timeout value in milliseconds.
  default: 100 ms.

- optionName=TCP_RTO_MAX: Size: 32 bit, sizeof(long)
  Set the maximum retransmission timeout value in milliseconds.
  default: 64000 ms.

- optionName=TCP_TS: Size: 16 bit, sizeof(int)
  Turn on/off the TCP time stamp option. 1: Enable TCP time stamp, 0: disable
  default: 1
  *Notes:*
  1. This value can not be changed after a connection is established.
  2. Also the socket must not be in listen mode.

**protocol_level=SOCKET_LEVEL**

- optionName=SO_REUSEADDR: Size: 16 bit, sizeof(int)
  Enable/disable local address reuse, 1: enable, 0: disable
  default: 1

- optionName=SO_KEEPALIVE: Size: 16 bit, sizeof(int)
  Keep connections alive, 1: enable, 0: disable
  default: 0

- optionName=SO_SNDBUF: Size: 32 bits, sizeof(long)
  Socket send buffer size. We recommend a maximum size of 8192 Bytes.
  default: TCP 4096, UDP 2048 bytes.

- optionName=SO_RCVBUF: Size: 32 bits, sizeof(long)
  Socket input buffer size. We recommend a maximum size of 8192 Bytes.
  default: TCP 4096, UDP 2048 bytes.

### Related Topics

API function **API_GETSOCKOPT** - Get socket options
**SetSocketOption** structure typedef

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## IcmpMib

```
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;
```

### SC1x Comments

These structures are only available in SC1x @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they will need access to these internal TCP/IP variables.

### Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Iface_Entry

```
typedef struct tag_iface_device
{
     unsigned int  devIndex;       // Internal index number
     char          devName[14];    // Device name, terminated by zero
     unsigned long devIPAddr;      // IP address for this interface
     unsigned long devNetmask;     // Netmask for the route
     unsigned long devDestIpAddr;  // Remote peer address for PPP
     unsigned char PhysAddr[6];    // Physical device address, max. 6 Bytes
     int           devType;        // Type of the device driver:
                                   // 0: unknown,
                                   // 1: Ethernet driver,
                                   // 2: PPP protocol
                                   // 4: SLIP protocol (not supported)
                                   // 5: Internal loopback
     int           devDHCP;        // Interface configured by DHCP? 1:0
     int           devFlag;        // Device status flag
                                   // Bit1 == 1  Device opened
                                   // Bit2 == 1  Device IP config in progress(PPP or DHCP)
                                   // Bit3 == 1  Device open completed
     int           devMTU;         // Max. Transfer Unit
     void far *    reserved;
} Iface_Entry;
```

### Related Topics

API function **GET_IFACE_ENTRIES** - Read table of TCP/IP device interfaces

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

# Iface_Entry_Ext

**SC1x:**

```
typedef struct tag_iface_device_ext
{
   unsigned int  devIndex;        //Internal index number
   char          devName[14];     //Device name
   unsigned long devIPAddr[4];    //IP addresses for this interface
   unsigned long devNetmask[4];   //Netmasks for the route
   unsigned long devDestIpAddr[4];//Remote peer addresses for PPP
   unsigned char PhysAddr[6];     //Physical device address, max. 6 Bytes

   int  devType;                  // type of the device driver:
                                  // 0: unknown,
                                  // 1: ethernet driver,
                                  // 2: use PPP protocol   server
                                  // 3: use PPP protocol client
                                  // 4: Internal loopback
   int  devDHCP[4];               // Interface configured by DHCP 1:0
   int  devFlag;
   int  devMTU;                   // Max. Transfer Unit
   void far *    IfaceHandle;     // handle type, to identify the interface internal
}Iface_Entry_Ext;
```

**SC1x3/SC23:**

```
typedef struct tag_iface_device_ext
{
  unsigned int  devIndex;        // Internal index number
  char          devName[14];     // Device name
  unsigned long devIPAddr[5];    // IP addresses for this interface
  unsigned long devNetmask[5];   // Netmasks for the route
  unsigned long devDestIpAddr[5];// Remote peer addresses for PPP
  unsigned char PhysAddr[6];     // Physical device address, max. 6 Bytes

  int  devType;                  // type of the device driver:
                                 // 0: unknown,
                                 // 1: ethernet driver,
                                 // 2: use PPP protocol   server
                                 // 3: use PPP protocol client
                                 // 4: Internal loopback
   int  devDHCP[5];              // Interface configured by DHCP 1:0
   int  devFlag;
   int  devMTU;                  // Max. Transfer Unit
   void far *    IfaceHandle;    // handle type, to identify the interface internal
}Iface_Entry_Ext;
```

## Related Topics

API function **GET_IFACE_ENTRIES** - Read table of TCP/IP device interfaces

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |



# IfaceIPv6_Entry

```
typedef struct tag_user_iface_ipv6_entry{
    void  far *     devIfaceHandle;       // Device handle pointer (internal used)
    int             devIndex;             // Internal used index number
    int             devType;              // Type of the device driver:
                                          // 0: unknown,
                                          // 1: Ethernet driver,
                                          // 2: PPP protocol
                                          // 4: SLIP protocol (not supported)
                                          // 5: Internal loopback
    char            devName[14];          // Device name, terminated by zero
    unsigned int    devFlags;             // Device status flag
    unsigned int    devIPv6Flags;         // IPv6 status flags
    unsigned int    devIPv6Mtu;           // The MTU of the IPv6 link
    struct in6_addr devIPv6PPPPeerIpAddr; // Remote peer address for PPP
    struct in6_addr devIPv6AddrArray[10];
    unsigned char   devIPv6PrefixLen[10];
    unsigned char   devIPv6AddrFlagsArray[10]; // Array of configuration flags
    unsigned char   devPhysAddr[6];       // Physical address of the interface
    void far      * devIPv6ExtensionPtr; //for future use
}IfaceIPv6_Entry;
```

## Comments

Further explanations for certain structure members follows.

**devFlags**
Device status flags:

Bit1 == 1 Device opened
Bit2 == 1 Device IPv6 config in progress
Bit3 == 1 Device open completed

**devIPv6Flags**
IPv6 status flags:

Bit1 == 1 Link-layer address configured
Bit2 == 1 Link-local address configured (or is pending)
Bit3 == 1 Site-local address configured (or is pending)
Bit4 == 1 Global scope IPv6 native address
Bit5 == 1 Global scope IPv4-compatible IPv6 address configured
Bit6 == 1 Global scope 6-to-4 IPv6 address
Bit7 == 1 IPv6 connectivity established on PPP interface

**devIPv6AddrArray**
Array of all configured IPv6 addresses on this interface.

**devIPv6PrefixLen**
Array of all configured Prefix length of all IPv6 addresses.

## Related Topics

API function **GET_IPV6_IFACE_ENTRIES** - Read table of TCP/IPv6 device interfaces

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# IfMib

```
typedef struct tagIfMib           // Interface table data
{
    long          ifIndex;        // index of this interface
    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
    unsigned long   devSpecific;
} IfMib;
```

### SC1x Comments

These structures are only available in SC1x @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to the internal TCP/IP SNMP variables. The SNMP MIB variables are not a part of our current official 6 @CHIP-RTOS versions. It's necessary to order directly a @CHIP-RTOS version which includes this feature.

### Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## in_addr

```
struct in_addr
{
    u_long s_addr; // 32 bit netid/hostid address in network byte order
};
```

### Related Topics

**sockaddr_in** data structure

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# in6_addr

128-bit IPv6-formatted address in network byte order

```
struct in6_addr
{
    union
    {
        unsigned char  ip6U8[16];
        unsigned short ip6U16[8];
        unsigned long  ip6U32[4];
    } ip6Addr;
};
```

## Comments

Mapped IPv4 addresses will appear in the `ip6Addr.ip6U32[3]` member.

## Related Topics

See **sockaddr_in6** structure

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

# IpMib

```
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;
```

### SC1x Comments

These structures are only available in SC1x @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to these internal TCP/IP variables.

### Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## IpUserCallbackInfo

A structure of this type is passed as an input parameter to an IP user callback handler.

```
struct IpUserCallbackInfo
{
   int              size;        // Size of this struct
   unsigned long    srcAddr;     // Source IP Address (in long format, already in correct byte order)
   unsigned long    destAddr;    // Destination IP Address (in long format, already in correct byte order)
   unsigned int     srcPort;     // Source Port (already in correct byte order)
   unsigned int     destPort;    // Destination Port (already in correct byte order)
   unsigned char    protocol;    // Protocol (see list below)
   int              fragmented;  // 0: it is an unfragmented package, 1: it is a fragment
   unsigned int     dataLength;  // Length of the data in the IP package
                                 //  (only available if package is not fragmented!)
   void far *       dataPtr;     // Pointer to the IP data
                                 //  (only available if package is not fragmented!)
   //Extended: Since SC12 RTOS 1.11 B, SC13 1.01B
   // Use the size parameter of this struct at your application, for detecting,
   // if these struct members are available
   void *           ipHdrPtr;    // Pointer to the IP header
   unsigned int     ipHdrLen;    // length of IP header
} IpCallbackUserInfo;
```

### Comments

Protocol types enumerated by the `protocol` member are coded as follows:

1 ICMP (*Internet Control Management Protocol*)
2 IGMP (*Internet Group Management Protocol*)
6 TCP (*Transmission Control Protocol*)
17 UDP (*User Datagram Protocol*)

Note: The IP callback function will be called before defragmenting fragmented IP packages. So if we receive a fragmented package, the IP callback will be called for every received fragment.

### SC1x3/SC23 Comments

At the current SC1x3/SC23 CHIP-RTOS version the IP callback features exists only for IPv4 packets.

### Related Topics

API function **IP_USER_CB** - Install IP callback function

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Packet_Count

```
typedef struct tag_cnt_packet
{
    unsigned int far * cnt_all_packets;      // count all incoming Ethernet packets
    unsigned int far * cnt_ip_packets;       // count incoming IP packets
    unsigned int far * cnt_arp_packets;      // count incoming ARP packets
    unsigned int far * cnt_tcp_packets;      // count incoming TCP packets
    unsigned int far * cnt_udp_packets;      // count incoming UDP packets
    unsigned int far * cnt_icmp_packets;     // count incoming ICMP packets

    unsigned int far * cnt_all_sended_packets;  // count all sent Ethernet packets
    unsigned int far * cnt_ip_sended_packets;   // count all sent IP packets
    unsigned int far * cnt_arp_sended_packets;  // count all sent ARP packets
    unsigned int far * cnt_tcp_sended_packets;  // count all sent TCP packets
    unsigned int far * cnt_udp_sended_packets;  // count all sent UDP packets
    unsigned int far * cnt_icmp_sended_packets; // count all sent ICMP packets

    unsigned int far * cnt_ip_chksum_errs;   // checksum errors on incoming IP packets
    unsigned int far * cnt_udp_chksum_errs;  // checksum errors on incoming UDP packets
    unsigned int far * cnt_tcp_chksum_errs;  // checksum errors on incoming TCP packets
    unsigned int far * cnt_eth_errs;         // errors on incoming Ethernet packets

} Packet_Count;
```

### Comments

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** - Access packet counts.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## PasCallBack

```
typedef struct tag_PasCallBack
{
    int         sd;     // socket descriptor
    int         event;  // occurred event
} PasCallBack;
```

### Comments

The pointer to this structure will be passed to a Pascal callback function in the registers ES:DI. You can read out the information about the socket and the event which has triggered the callback function.

### Related Topics

To register a Pascal callback function see **API_REGISTER_CALLBACK_PASCAL**

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | n/a | n/a |

**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 far *remoteHostNamePtr; // Remote IP
   int     pingInterval;       // seconds
   int     pingDataLength;     // Maximum 2080 bytes
   unsigned long count;        // Limit number of pings sent.  Set to
                               //  zero if ping should run forever (until PING_CLOSE)

   unsigned char pingstate;    // ping socket state, 1: open 0: closed
   // Statistics, filled in by system inside PING_STATISTICS API:
   unsigned long transmitted;  // Sent ping requests count
   unsigned long received;     // Received replies count
   unsigned int lastsenderr;   // Last send error
   unsigned int lastrcverr;    // Last receive error
   unsigned long maxRtt;       // Maximum round trip time (ms),
                               //  rounded off to 100 ms step
   unsigned long minRtt;       // Minimum round trip time (ms), (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 in seconds
`pingDataLength`... size of ping data blocks
`count`... set to zero, if ping should run forever

The remainder of the data structure is managed within the API functions.

### Related Topics

API function **PING_OPEN** - Start ICMP echo requests

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.21 | V1.21 | V1.21 | V1.10 | V1.00 |

Top of list
Index page

# PKI_CERT_ADD_Entry

```
typedef struct tag_pki_cert_add_entry
{
  void far * fileNamePtr; // file name of certificate
  int        fileFormat;  // file format:      0x01 = PEM format,
                          //                   0x02 = DER format
  int        certType;    // certificate type: Bit0 = 1: local certificate,
                          //                   Bit1 = 1: non local certificate (CA or peer),
                          //                   Bit3 = 1: root CA certificate (use together with Bit1)
  void far * idPtr;       // identification string
  int        idLength;    // length of identification string
  void far * caIdPtr;     // CA identication string
} PKI_CERT_ADD_Entry;
```

## Comments

If caIdPtr is NULL then the PKI will automatically search the existing certificate database to find the CA by matching DN (distinguished name). If the root CA is added caIdPtr should be NULL.

## Related Topics

API function **PKI_CERT_ADD** Add certificate to public key infrastructure

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

# PKI_OWNKEYPAIR_ADD_Entry

```
typedef struct tag_pki_ownkeypair_add_entry
{
  void far * fileNamePtr; // file name of public/private key file
  int        fileFormat;  // file format: 0x01 = PEM format,
                          //              0x02 = DER format
  int        keyType;     // key type:    0x00 = RSA key
} PKI_OWNKEYPAIR_ADD_Entry;
```

## Related Topics

API function **PKI_OWNKEYPAIR_ADD** Add a public/private key pair

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

## PPPClie_ModemCmd

```
typedef struct tag_pppclie_modem_cmd
{
      char  * modemcmd;    //modem command string (e.g. +++ or ATO);
      char  * modemans;    //expected answer from modem (NULL for no answer expected);
      int   timeout;       //Max. Timeout in seconds to wait for an answer
      int   retries;       //Number of retries
}PPPClie_ModemCmd;
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.26 | V1.26 | V1.26 | V1.17 | V1.17 |

Top of list
Index page

## PPPClie_ModemCtrl

```
typedef struct tag_pppclie_modem_ctrl
{
   //idle interval time, at which the pppclient executes the configured control commands
   int ctrl_idle_secs;
   //modem control commands to execute
   PPPClie_ModemCmd ctrlcmd[3];
}PPPClie_ModemCtrl;


Example for a modem control variable:

PPPClie_ModemCtrl modem_ctrl =
{  20,                // idle interval after which the modem commands are executed
   {{"+++","OK",2,2}, // Switch into modem command mode
                      // If the modem is not connected it doesn't answer with "OK"
                      // -> close ppp session
    {"AT","OK",2,2},  // Check, if the modem is in command mode
    {"ATO",NULL,0,0}  // Switch modem back to online mode
   }
};
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.26 | V1.26 | V1.26 | V1.17 | V1.17 |

Top of list
Index page

## PPPClient_Init

```
typedef struct tag_ppp_client
{
   int               port;
   int               auth;
   int               modem;
   int               flow;
   long              baud;
   unsigned long     idletimeout;
   char              username[PPP_MAXNAME_LEN];
   char              password[PPP_MAXNAME_LEN];
   void              far * dptr;

   char              PPPClieipAddrStr[16];
   char              PPPClieRemipAddrStr[16];
   char              PPPClienetMaskStr[16];
   char              PPPClieipGatewayStr[16];

   PPPDial           pppdial[PPP_MAX_DIAL];
   PPP_ModemHangup   modem_hangup;
   int               break_modem;
} PPPClient_Init;
```

## Comments

The `PPPClient_Init` structure is used to open a PPP client session. For how to initialize and use these structures see the PPPCLIE.C example. The structure has the following members:

**`auth`**

0: no authentication
1: PAP Client must send user name and password for PAP authentication to the peer
2: CHAP Client must send user name and password for CHAP authentication to the peer
3: PAP Client expects PAP user name and passwordfrom the peer
4: CHAP Client expects CHAP user name and password from the peer

**`modem`**

modem usage:
0: null modem, 1: modem
2: modem usage with control check enabled (in idle case)
After each time interval during which the PPP client receive no data, the PPP client executes cyclic **in intervals of a defined idle time**an user defined set of modem commands to check the status of the modem. The PPP client closes the connection, if the execution of one of these commands failed. The idle interval and the modem command set must be specified at a variable of type **PPPClie_ModemCtrl**. The PPP client expects the address of this variable at the pointer **dptr**of the PPPClient_Init structure variable.
3: modem usage with cyclic control check enabled
Same as modem == 2 with one difference:
After each time interval the client executes the configured modem commands. In this mode the configured modem commands are not printed via stdout at execution time, even if **MODEMTRACE**is enabled.

**`baud`**

Serial port baud rate

**`idletimeout`**

Close PPP after this amount of idle time, in seconds.
(0: No closing after idle time)

**`username`**

Applicable if `auth`is non-zero.

**`password`**

Applicable if `auth`is non-zero.

**`dptr`**

If parameter modem==2 (see above), pointer to a variable of type **PPPClie_ModemCtrl**
else: Not used.

**`PPPClieipAddrStr`**

If this IP string is set to "0.0.0.0" then the client expects an IP address from the peer. This IP string is then updated after a successful connection is established.

If this string not set to "0.0.0.0" then the client will use this IP address during the PPP session.

**`PPPClieRemipAddrStr`**

If this IP string is set to "0.0.0.0" then the client allows the peer to use its own IP address during the PPP session. This string is then updated after a successful connection has been established.

If this IP string is not set to "0.0.0.0" then the client will configure the remote peer with this IP address.

**PPPClienetMaskStr**
Subnet mask.

**PPPClieipGatewayStr**
Gateway IP address.

**pppdial**
This array of the PPPdial**structures**is used for initializing and dialing a modem. These modem commands will be executed at the start of establishing a connection to a PPP server.

**modem_hangup**
This PPP_ModemHangup**structure**is used when closing the modem connection.

**break_modem**
Flag for interrupting IPC@CHIP®-to-modem communication, break dialing, connect control communication (dialing, waiting for connect). Setting break_modemto 1 breaks current modem control communication between IPC@CHIP®and the modem at a PPP client open or close call. The PPP client reads this flag and breaks the dialing if the flag is set. This flag can be set from another task. It will not break an established PPP link! Don't forget to clear this flag to zero after breaking.

## SC1x3/SC23 Comments

**port**
Serial port (0:EXT 1:COM 2:SER2 3:SER3)

**flow**
Serial port flow control: (0: none, 1:XON/XOFF, 2:RTS/CTS)

The flow control mode XON/XOFF applied to PPP has not been tested. It is not advisable to use it. The XON/XOFF mode is also available if the **DMA**receive mode for the selected serial port is enabled, but due to the internal functionality of DMA it is not possible to immediately detect an XON or XOFF from the peer. Therefore it is possible that an overrun situation can occur at the connected peer (e.g. GSM modem). We now offer this mode because some GSM modems support only XON/XOFF flow control.

## SC1x Comments

**port**
Serial port (0:EXT 1:COM)

**flow**
Serial port flow control: (0: none, 1:XON/XOFF, 2:RTS/CTS)

The flow control mode XON/XOFF applied to PPP has not been tested. It is not advisable to use it. Since @CHIP-RTOS 1.02B XON/XOFF mode is also available if the **DMA**receive mode for the selected serial port is enabled, but due to the internal functionality of DMA it is not possible to immediately detect an XON or XOFF from the peer. Therefore it is possible that an overrun situation can occur at the connected peer (e.g. GSM modem). We now offer this mode because some GSM modems support only XON/XOFF flow control.

## Related Topics

API function **PPPCLIENT_OPEN** - Open PPP client session.
**PPPdial** dial-up command data

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# PPPClient_Init_IPv6

```
typedef struct tag_ppp_client_ipv6
{
   int               port;
   int               auth;
   int               modem;
   int               flow;
   long              baud;
   unsigned long     idletimeout;
   char              username[PPP_MAXNAME_LEN];  // Used if .auth != 0
   char              password[PPP_MAXNAME_LEN];  // Used if .auth != 0
   void       far  * dptr;                     // dummy ptr
   char              PPPClieipAddrStr[INET6_ADDRSTRLEN];
   char              PPPClieRemipAddrStr[INET6_ADDRSTRLEN];
   PPPDial           pppdial[PPPCLIE_MAX_DIAL];
   PPP_ModemHangup   modem_hangup;
   int        far   * break_modem;

}PPPClient_Init_IPv6;
```

## Comments

The `PPPClient_Init_IPv6` structure is used to open a PPP client session. For how to initialize and use these structures see the PPPCLIE.C example. The structure has the following members:

**`port`**
Serial port (0:EXT 1:COM 2:SER2 3:SER3)

**`auth`**
0: no authentication
1: PAP Client must send user name and password for PAP authentication to the peer
2: CHAP Client must send user name and password for CHAP authentication to the peer
3: PAP Client expects PAP user name and passwordfrom the peer
4: CHAP Client expects CHAP user name and password from the peer

**`modem`**
modem usage (0:null modem, 1:modem)

**`flow`**
Serial port flow control: (0: none, 1:XON/XOFF, 2:RTS/CTS)

**`baud`**
serial baud rate

**`idletimeout`**
Close PPP after this amount of idle time, in seconds.
(0: No closing after idle time)

**`username`**
Applicable if `auth`is non-zero.

**`password`**
Applicable if `auth`is non-zero.

**`dptr`**
Not used.

**`PPPClieipAddrStr`**
Storage for auto configured IPv6 address string. This will be filled in after the PPP link has been established.

**`PPPClieRemipAddrStr`**
Storage for auto configured Remote peer IPv6 address string. This will be filled in after the PPP link has been established.

**`pppdial`**
This array of the `PPPdial`**structures**is used for initializing and dialing a modem. These modem commands will be executed at the start of establishing a connection to a PPP server.

**`modem_hangup`**
This `PPP_ModemHangup`**structure**is used when closing the modem connection.

**`break_modem`**
Pointer to a flag for interrupting IPC@CHIP®-to-modem communication, break dialing, connect control communication (dialing, waiting for connect). Setting the flag referenced by `break_modem`pointer to 1 breaks current modem control communication between IPC@CHIP®and the modem at a PPP client open or close call. The PPP client reads this flag

and breaks the dialing if the flag is set.  This flag can be set from another task.  It will not break an established PPP link!  Don't forget to clear this flag to zero after breaking.

### Related Topics

API function **PPPCLIENT_OPEN_IPV6** - Open PPP client session.
**PPPdial** dial-up command data

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## PPPDial

```
typedef struct tag_pppdial_init
{
      char far * modemcmd;  // modem command string
      char far * modemans;  // modem answer  string (max. 80 characters + '\0')
      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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## PPP_IPCfg_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_IPCfg_Data;
```

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## PPP_IPCfg_Data_IPv6

```
typedef struct tag_pppipcfg_data_ipv6
{
     char          IP[46];          // auto configured PPP server IP
     char          RemIP[46];       // auto configured Remote peer IP
     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_IPCfg_Data_IPv6;
```

### Comments

The `PPP_IPCfg_Data_IPv6` structure is used to read out the configuration of the PPP server.

### Related Topics

API function **PPPSERVER_GET_CFG_IPv6** - Get PPP server IPv6 configuration

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## PPP_ModemHangup

```
typedef struct tag_pppclie_hangup
{
      char far *modemcmdmode;    // string for switching modem into command mode e.g. +++
      int      delay;            // delay time after switching in seconds
      PPPDial pppdial[PPP_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

**Top of list**
**Index page**

---

## PPP_Option

This data structure and associated constants are defined in TCPIPAPI.H header file. It is used to control PPP options for both the PPP server and client.

```
typedef struct tag_ppp_option
{
   int             protocolLevel;     // PPP_LCP_PROTOCOL or PPP_IPCP_PROTOCOL
   int             remoteLocalFlag;   // PPP_OPTION_WANT or PPP_OPTION_ALLOW
   int             optionName;        // From constants defined below
   const char far *optionValuePtr;    // To buffer provided by user
   int             optionLength;      // Number of bytes at *optionValuePtr
} PPP_Option;

 // PPP Protocol levels
 #define PPP_LCP_PROTOCOL     0x21c0  // LCP,  Link control protocol
 #define PPP_IPCP_PROTOCOL    0x2180  // IPCP, Internet protocol control protocol
 #define PPP_PROTOCOL         0x0000  // PPP common

 // For .remoteLocalFlag
 #define PPP_OPTION_WANT           0  // What we want to use for our side
 #define PPP_OPTION_ALLOW          1  // What we will allow the remote side to use

 // Protocol options for .optionName field
 // LCP protocol options
 #define PPP_LCP_MAX_RECV_UNIT     1  // Maximum receive unit
 #define PPP_LCP_ACCM              2  // Async Ctrl Char Map
 #define PPP_LCP_AUTH_PROTOCOL     3  // Authentication, def. 0, else PPP_PAP or PPP_CHAP
 #define PPP_LCP_PROTOCOL_COMP     7  // Turn On/Off protocol field compression
 #define PPP_LCP_ADDRCONTROL_COMP  8  // Turn ON/OFF address/control field compression
 #define PPP_LCP_TERM_RETRY        9  // Terminate retries,
 #define PPP_LCP_CONFIG_RETRY      10 // Config retries,
 #define PPP_LCP_TIMEOUT           11 // Retransmission timeout in seconds
 #define PPP_LCP_MAX_FAILURES      12 // LCP maximum failure

 // IPCP protocol options
 #define PPP_IPCP_COMP_PROTOCOL    2  // VJ-TCP/IP header compression
 #define PPP_IPCP_IP_ADDRESS       3  // IP Address the requesting system wishes to use
 #define PPP_IPCP_VJ_SLOTS         4  // Number of VJ slots
 #define PPP_IPCP_RETRY            5  // IPCP Timeout, retry options
 #define PPP_IPCP_TIMEOUT          6  // IPCP timeout seconds
 #define PPP_IPCP_MAX_FAILURES     7  // LCP maximum failures
 #define PPP_IPCP_DNS_PRI         29  // Specify DNS prim. server IP
 #define PPP_IPCP_DNS_SEC         31  // Specify DNS sec.  server IP

 //PPP options
 #define PPP_SEND_BUFFER_SIZE      1  // Data buffered by PPP link layer,
                                      // before the device driver send function
                                      // is called, length 2, def. 1 (Byte)

 //Authentication constants
 #define PPP_PAP              0x23c0  // PAP authentication
 #define PPP_CHAP             0x23c2  // CHAP authentication
```

## Comments

The type of data found at the location referenced by the `optionValue` member varies depending on the particular PPP 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 (in bytes) are stated at the descriptions list below. The parameters `optionValue` and `optionLength` are used to access option values.

The structure member `remoteLocalFlag` can have two possible values:
`PPP_OPTION_WANT` or `PPP_OPTION_ALLOW`.

With `remoteLocalFlag` set to `PPP_OPTION_WANT`, the IPC@CHIP® sends a PPP configuration request to the connected peer which contains the specified option.

With `remoteLocalFlag` set to `PPP_OPTION_ALLOW`, the IPC@CHIP® accepts an incoming PPP configuration request from the connected peer for the specified option.

The following list specifies the PPP options for the two different protocol levels, including a short description and the length of the option needed for setting the `optionLength` member. Protocol level and `optionName` constants referred to here are defined in TCPIPAPI.H

**`protocolLevel=PPP_LCP_PROTOCOL`** (*Link Control Protocol* level options)

- `optionName=PPP_LCP_ADDRCONTROL_COMP`: Size: 8 bit, sizeof(char) - Activate/Deactivate address control compression, default 0: Deactivated
- `optionName=PPP_LCP_PROTO_COMP`: Size: 8 bit, sizeof(char) - Value 1: Protocol field compression, default 0: No compression
- `optionName=PPP_LCP_ACCM`: Size: 32 bit, sizeof(long) - Set Async control character map. Each bit position of the unsigned long value represents the corresponding character which should be escaped or not during PPP session.
- `optionName=PPP_LCP_MAX_RECV_UNIT`: Size: 16 bit, sizeof(int) - Maximum receive unit, default 1500 (bytes)
- `optionName=PPP_LCP_AUTH_PROTOCOL`: Size: 16 bit, sizeof(int) - Authentication method, default 0 (no Auth.), possible PPP_PAP or PPP_CHAP
- `optionName=PPP_LCP_TERM_RETRY` : Size: 8 bit, sizeof(char) - Terminate retries, default 3
- `optionName=PPP_LCP_CONFIG_RETRY` : Size: 8 bit, sizeof(char) - Config retries, default 10
- `optionName=PPP_LCP_TIMEOUT`: Size: 8 bit, sizeof(char) - Retransmission timeout in seconds, default 3 (seconds)
- `optionName=PPP_LCP_MAX_FAILURES` : Size: 8 bit, sizeof(char) - LCP maximum failures ,default 5

**`protocolLevel=PPP_IPCP_PROTOCOL`** (IpCp level options)

- `optionName=PPP_IPCP_COMP_PROTOCOL`: Size: 16 bit, sizeof(int) - Value 1: VJ TCP/IP header compression, default 0: No compression
- `optionName=PPP_IPCP_DNS_PRI`: Size: 32 bit, sizeof(long) - Specifies the IP addresses of the primary DNS server we will allow the remote to use or the primary DNS server we want to use. If this option is called with the value 0 and `remotelocalflag==PPP_OPTION_WANT`, then we allow the connected peer to report their primary DNS server IP.
- `optionName=PPP_IPCP_DNS_SEC`: Size: 32 bit, sizeof(long) - Specifies the IP addresses of the secondary DNS server we will allow the remote to use or the secondary DNS server we want to use. If this option is called with the value 0 and `remotelocalflag==PPP_OPTION_WANT`, then we allow the connected peer to report their primary DNS server IP.
- `optionName=PPP_IPCP_IP_ADDRESS`: Size: 32 bit, sizeof(long) - The IP Address the requesting system wishes to use, default 0 (not allowed) By default the remote is not allowed to request its IP and/or DNS IP addresses If PPPSetOption is called with OPTION_ALLOW, and value 0, the remote will be allowed to set its addresses.
- `optionName=PPP_IPCP_VJ_SLOTS` : Size: 1 bit, sizeof(char) - Number of VJ slots, default 1 (slot)
- `optionName=PPP_IPCP_RETRY`: Size: 1 bit, sizeof(char) - IPCP Timeout, retry options, default 10
- `optionName=PPP_IPCP_TIMEOUT`: Size: 1 bit, sizeof(char) - IPCP timeout seconds, default 1 (second)
- `optionName=PPP_IPCP_MAX_FAILURES` : Size: 1 bit, sizeof(char) - LCP maximum failures, def. 5

If the option PPP_IPCP_PROTOCOL (VJ TCP/IP header compression) is set, it can be possible that the FTP server of the IPC@CHIP® is not usable via the PPP interface.

## Related Topics

API function **PPPCLIENT_SET_OPTIONS** - Set PPP client options
API function **PPPSERVER_SET_OPTIONS** - Set PPP server options
API function **PPPCLIENT_GET_DNSIP** - Get DNS IP
API function **DEV_PPP_SET_OPTION** - Set PPP option at own device drivers

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

# recv_params

```
struct recv_params
{
   char          far      *bufferPtr;        // Store incoming data here
   int                     bufferLength;     // Maximum bytes to store
   int                     flags;            // Blocking, timeout or no wait
   struct sockaddr  far    *fromPtr;         // Only needed for UDP.
   int              far    *fromlengthPtr;   // Only needed for UDP.
   unsigned long           timeout;          // timeout milliseconds
};
```

## Comments

This structure is defined in tcpip.h.

**bufferPtr**
This output parameter specifies the user buffer into which the received data will be placed by the system.

**bufferLength**

Size of user buffer at `bufferPtr`, places upper limit on bytes transferred.

**flags**

Set to one of following constants defined in tcpip.h:
MSG_BLOCKING: Sleep until data comes in
MSG_TIMEOUT : The caller wakes up after timeout or if data comes in. The structure member `timeout`must be set.
MSG_DONTWAIT: Return immediately

**fromlengthPtr** UDP only

The 16 bit value referenced by this pointer is both an input and output parameter, which should be preset to:

```
sizeof(struct sockaddr_in)
```

prior to calling **API_RECVFROM**.

**timeout**
If `flags`is set to MSG_TIMEOUT, then the maximum number of milliseconds to wait should be specified here. The timeout resolution is 100 milliseconds.

## SC1x Comments

**fromPtr** UDP only
This is an output parameter. The referenced `sockaddr`**structure**is cast to a `sockaddr_in`**structure**prior to usage. This field will be set by the system inside the **API_RECVFROM**call to indicate the origin of the received UDP data.

## SC1x3/SC23 Comments

**fromPtr** UDP only

This is an output parameter. The referenced sockaddr **sockaddr structure** is cast to a **sockaddr_in** structure (AF_INET IPv4 sockets)
or to a **sockaddr_in6** structure (AF_INET6 IPv6 sockets) prior to usage.

This field will be set by the system inside the **API_RECVFROM** call to indicate the origin of the received UDP data.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**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 for this route
     int hops;                     // Number of routers between this host and route
} Route_Entry;
```

### SC1x3/SC23 Comments

At the current SC1x3/SC23 CHIP-RTOS version routing functions are existing only for IPv4 protocol.

### Related Topics

API function **ADD_STATIC_ROUTE** - Add a route for an interface
API function **DEL_STATIC_ROUTE** - Delete a route from an interface

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## send_params

```
struct send_params
{
   char             far   *bufferPtr;        // Pointer to send data
   int                     bufferLength;     // Number of bytes to send
   int                     flags;            // Blocking or no wait
   struct sockaddr  far   *toPtr;            // only needed for UDP
   int              far   *tolengthPtr;      // only needed for UDP
};
```

## Comments

This structure is defined in tcpip.h. The elements are used as follows:

**`bufferPtr`**
Pointer to data to be sent.

**`bufferLength`**
Number of bytes to be sent from `bufferPtr`.

**`flags`**
Set to one of following constants defined in tcpip.h:
MSG_BLOCKING: Sleep until data is transferred into the socket transmit queue
MSG_DONTWAIT: Return immediately after loading bytes into available transmit queue space

**`tolengthPtr`** UDP only

The 16 bit value referenced by this pointer must be: `sizeof(struct sockaddr_in)`

## SC1x Comments

**`toPtr`** UDP only
The referenced structure is cast to a `sockaddr_in`**structure**prior to usage. The datagram destination IP address and port number is specified here by caller.

## SC1x3/SC23 Comments

**`toPtr`** UDP only
The referenced structure is cast to a **sockaddr_in**structure (AF_INET IPv4 sockets) or to a **sockaddr_in6**structure (AF_INET6 IPv6 sockets) prior to usage.
The datagram destination IP address and port number is specified here by caller.

## Related Topics

API function **API_OPENSOCKET_IPV6** - Open IPv6 socket
API function **API_OPENSOCKET** - Open IPv4 socket

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# SetSocketOption

This data structure is defined in the TCPIPAPI.H library header file.

```
typedef struct tag_setsockopt
{
    int     protocol_level;      // Protocol level: IP level, TCP level or socket level
    int         optionName;      // Option's name constant
    const char far *optionValue; // Pointer to the option value (type varies)
    int        optionLength;     // Length of option value data

} SetSocketOption;

 // Example usage of SetSocketOption type to set
 // IP Time-to-Live to 69 seconds (or router hops)
 // using C-Library function setsockopt():

 unsigned char time_to_live = 69;
 int error ;
 int socketdescriptor;  // You must initialize this (not shown here)
 SetSocketOption sockopt = {IP_PROTOIP_LEVEL,            // .protocol_level
                            IPO_TTL,                     // .optionName
                       (const char far *)&time_to_live,  // .optionValue
                            sizeof(unsigned char)};      // .optionLength

 setsockopt (socketdescriptor, &sockopt &error) ;
```

### 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

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## sockaddr

```
struct sockaddr
{
    unsigned char  sa_len;      // Total Length
    unsigned char  sa_family;   // Address Family AF_xxx
    char           sa_data[14]; // up to 14 bytes of protocol specific stuff
};
```

### Comments

This generic "one size fits all" BSD structure is treated as a `sockaddr_in` **structure** within the TCP/IP API functions.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**

# sockaddr_in

Socket data structure

**SC1x:**

```
struct sockaddr_in
{
   short           sin_family;  // AF_INET
   unsigned short  sin_port;    // 16 bit Port Number in network byte order
   struct  in_addr sin_addr;    // 32 bit netid/hostid in network byte order
   char            sin_zero[8]; // unused
};
```

**SC1x3/SC23:**

```
struct sockaddr_in
{
  unsigned char   sin_len;     //
  unsigned char   sin_family;  // AF_INET
  unsigned int    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.

The htons function can be used to convert port numbers to network byte order.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

# sockaddr_in6

Data structure required by the 4.4BSD socket API used to represent an IPv6 address

```
struct sockaddr_in6
{
    unsigned char    sin6_len;       //Length (in bytes) of the sockaddr_in6 structure
    unsigned char    sin6_family;    //Address family, set to AF_INET6
    unsigned short   sin6_port;      //Transport layer port number
    unsigned long    sin6_flowinfo;  //IPv6 packet header Flow Label
    struct in6_addr  sin6_addr;      //128-bit IPv6-formatted address in network byte order.
    unsigned long    sin6_scope_id;  //sin6_scope_id for local scope
                                     // (i.e. link-local, site-local)
                                     // unicast IPv6 addresses
};
```

### Comments

The `sin6_len` is the length in bytes of struct sockaddr_in6.

The `sin6_family` member should be set to `AF_INET6`.
Note that an IPv4-only IP address can be represented as a 128-bit IPv6-formatted IP Address.
The socket API refers to this as an IPv4-mapped IPv6 address. In this case, sin6_family is set to AF_INET6, even though the structure contains an IPv4-only IP Address.
However, internally the stack treats it as an IPv4-only IP address.

The `sin6_flowinfo` member contains the IPv6 packet header Flow Label as the low-order 20 bits and Traffic Class as the next more significant 8 bits. The high-order 4 bits of this field are reserved. This field allows the application to set the flow label and traffic class associated with the socket via a call to bind or connect.

### Related Topics

See **in6_addr** structure
See **sockaddr_storage** structure
See **sockaddr_in** structure

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## sockaddr_storage

Because the sockaddr type isn't big enough to hold an IPv6 address, the sockaddr_storage type is provided which can hold either an IPv4 or an IPv6 address.

```
struct sockaddr_storage
{
    union
    {
        struct sockaddr_in6     ipv6;
        struct sockaddr_in      ipv4;
    } addr;
};
```

### Related Topics

See **sockaddr_in6** structure
See **sockaddr_in** structure

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## SocketInfo

```
typedef struct tag_socket_info{
        unsigned int  socIndex;
        unsigned char protocol;        // 6: TCP   17: UDP
        unsigned int  localPort;       // Little endian!
        unsigned long IfIpAddress;     // network byte order
        unsigned int  remotePort;      // Little endian!
        unsigned long remoteIP;        // network byte order
        unsigned char tcpState;
} SocketInfo;
```

### Comments

Note that the `localPort` and `remotePort` members are in little endian format, while the two IP addresses remain in network byte order (big endian).

Possible TCP socket states:

0 CLOSED
1 LISTEN
2 SYN_SENT
3 SYN_RECEIVED
4 ESTABLISHED
5 CLOSE_WAIT
6 FIN_WAIT_1
7 CLOSING
8 LAST_ACK
9 FIN_WAIT_2
10 TIME_WAIT
20 INVALID

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## SocketInfoIpv4v6

```
typedef struct tag_socket_info{
        unsigned int  socIndex;
        unsigned char protocol;        // 6: TCP   17: UDP
        unsigned int  localPort;       // Little endian!
        struct in6_addr IfIpAddress;   // network byte order
        unsigned int  remotePort;      // Little endian!
        struct in6_addr remoteIP;      // network byte order
        unsigned char tcpState;
} SocketInfo;
```

### Comments

Note that the `localPort` and `remotePort` members are in little endian format, while the two IP addresses remain in network byte order (big endian).

Possible TCP socket states:

0 CLOSED
1 LISTEN
2 SYN_SENT
3 SYN_RECEIVED
4 ESTABLISHED
5 CLOSE_WAIT
6 FIN_WAIT_1

7 CLOSING
8 LAST_ACK
9 FIN_WAIT_2
10 TIME_WAIT
20 INVALID

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## SSL_SESSION_NEW_Entry

```
typedef struct tag_ssl_session_new_entry
{
  char far *    certIdentity; // SSL server/client own certificate name
  unsigned int  maxCaches;    // Maximum number of caches for this session
  int           version;      // The SSL version number for this session
                              // Bit0: SSL 3.0
                              // Bit1: TLS 1.0
} SSL_SESSION_NEW_Entry;
```

### Comments

If the value in maxCaches is zero, there is no cache at all. That means there is no connection resumption at all either. Note that each cache entry will consume about 100 bytes memory.

### Related Topics

API function **SSL_SESSION_NEW** Open a new SSL session

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

Top of list
Index page

---

## TcpMib

```
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;
```

### SC1x Comments

These structures are only available in SC1x @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to the internal TCP/IP SNMP variables.

### Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## timeval

```
struct timeval {
  long    tv_sec;         // seconds
  long    tv_usec;        // and microseconds
};
```

### Comments

This data structure is used to specify the maximum wait time for the **select()** API. Set both members to zero for polling. The @Chip-RTOS timing resolution within the `select()` is to one millisecond. Nevertheless, the `tv_usec` is specified in microseconds here for consistancy with the Berkely Sockets standard.

Inside the @Chip-RTOS this all is going to be converted to a 31 bit number of milliseconds. So values for `tv_sec` greater than (2**31)/1000 seconds (= ~24 days) will overflow and therefore not provide your requested wait time.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

Top of list
Index page

# UdpMib

```
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;
```

## SC1x Comments

These structures are only available in SC1x @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS. But if a user is able to implement an agent based on the TCP/IP API, they need access to the internal TCP/IP SNMP variables.

## Related Topics

API function **API_SNMP_GET** - Get internal TCP/IP SNMP variables

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

# UserEthDhcp_Entry

```
typedef struct tag_UserDhcpEthEntry
{
    unsigned long   BootSIpAddress;      // BOOT Server (TFTP server), not supported
    unsigned long   Dns1ServerIpAddress; // Domain name server
    unsigned long   Dns2ServerIpAddress; // Second domain name server
    unsigned long   Yiaddr;              // Our (leased) IP address
    unsigned long   NetMask;             // Our subnet mask
    unsigned long   DefRouter;           // Default router
    unsigned long   DhcpServerId;        // DHCP selected server IP address
    unsigned long   internal1;           // Internal use only
    unsigned long   internal2;           // Internal use only
    unsigned long   DhcpLeaseTime;       // DHCP Address lease time in milliseconds
    unsigned long   internal3;           // internal use only
    unsigned long   internal4;           // internal use only
    unsigned char   DomainName[64];      // Domain name
    unsigned char   BootSname[64];       // TFTP server name, not used
    unsigned char   BootFileName[128];   // Boot file name (for TFTP download),
                                         //   not supported.
    unsigned short  BootFileSize;        // Boot file size in 512 Bytes blocks
                                         //    (for TFTP download).
    unsigned short  internal5;           // internal use only
} UserEthDhcp_Entry;
```

## Comments

Members of this structure hold the DHCP configuration data if the IPC@CHIP® is configured by a DHCP server.

This is read only information!

## Related Topics

API function **DHCP_STAT** - Get DHCP status of the internal Ethernet interface of the IPC@CHIP®
API function **DEV_WAIT_DHCP_COMPLETE** - Get DHCP status for a user Ethernet interface

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

End of document




# @CHIP-RTOS C Library - TCP/IP API

Header File: `#include <TCPIP.H>`
which includes `<TCPIPAPI.H>`

TCP/IP

IPSec/PKI/SSL

PPP Client

PPP Server

Other

## TCP/IP Functions

| | |
|---|---|
| Get_IPConfig | Get IP, subnet mask and/or gateway |
| Set_IPConfig | Set IP, subnet mask and/or gateway |
| PPP_Server_GetIP | Get IP Address of the PPP Server |
| PPP_Client_GetIP | Get IP Address of the PPP Client |
| InetToAscii | Convert an IPv4 address to an IP string |
| inet_addr | Converts a string containing an IPv4 address in dot notation |
| inet_pton | Convert an IP address (IPv4 or IPv6) from standard text representation to binary |
| inet_ntop | Convert an IP address (IPv4 or IPv6) from binary to standard text representation |
| --- Socket Functions --- | |
| opensocket | Open TCP or UDP IPv4 socket |
| opensocket_ext | Open an IPv4 or IPv6 socket |
| closesocket | Close socket |
| bind | Associates a local address with a socket |
| connect | Establish a connection to a specifed socket |
| GetWaitingBytes | Report the number of bytes on a socket |
| Set_IP_Type_Of_Service | Set socket's Type-Of-Service |
| setsockopt | Set options on a socket |
| getsockopt | Get options on a socket |

| | |
|---|---|
| Set_Blocking_Mode | Set a socket in blocking or non-blocking mode |
| get_socketerror | Get last socket error |
| select | Await events on a selected set of sockets |
| --- TCP/IP Sockets --- | |
| tcp_connect | Open TCP/IP socket and connect |
| accept | Accept the next incoming connection |
| listen | Listen for incoming connections |
| htons | Converts a 16 bit word from host to TCP/IP network byte order |
| send | Transmit a message (TCP) |
| recv | Receive message (TCP) |
| ResetConnection | Abort a connection on a socket |
| setlinger | Set linger time on close (TCP) |
| setreuse | Set reuse option on a listening socket |
| Get_TCP_Socket_State | Find TCP socket and return state |
| Get_TCP_Socket_State_IP_All | Find TCP socket and return state (IPv4 and IPv6) |
| FindAllOpenSockets | Get information about all open sockets |
| FindAllOpenSockets_IP_All | Get information about all open sockets (IPv4 and IPv6) |
| Shutdown | Shutdown socket in read,write or both directions |
| --- UDP Sockets --- | |
| recvfrom | Receive a datagram and store the source address |
| recvfromto | Receive a datagram and store source and dest. address |
| sendto | Transmit a datagram |
| sendto_iface | Transmit a datagram via specified device |
| --- DNS Resolver --- | |
| getHostByName | Get a host's address(es) by domain name (IPv4) |
| getHostByName2 | Get a host's address(es) by domain name (IPv6) |
| getHostByAddr | Get a host's domain name by address (IPv4) |
| getHostByAddr2 | Get a host's domain name by address (IPv6) |
| getMailHosts | Get mail hosts |
| setNameServer | Add prioritised name server |

## IPsec/PKI/SSL/CRYPT Functions

| | |
|---|---|
| --- IP Security --- | |
| IP Security | API |
| --- PKI Functions --- | |
| PKI_CertificateAdd | Add a certificate to the PKI |
| PKI_CertificateDel | Delete a certificate from the PKI |
| PKI_OwnKeyPairAdd | Add own private/public key pair |
| --- SSL Functions --- | |
| SSL_SessionNew | Open a new SSL session |

| | |
|---|---|
| SSL_SessionClose | Close a SSL session |
| SSL_SessionProp | Set SSL session proposals |
| SSL_SessionOpt | Set SSL session options |
| SSL_AddClientCA | Add a client CA certificate |
| SSL_ClientStart | Start the SSL client |
| SSL_SendFlush | Flush SSL user data |
| SSL_ConnClose | Close a SSL connection |
| SSL_Get_ConnState | Get SSL connection state information |
| SSL_HandshakeComplete | Check handshake status |
| --- Cryptographic Functions --- | |
| Crypt_MD5_Init | Init MD5 checksum generator |
| Crypt_MD5_Update | Update MD5 hash |
| Crypt_MD5_Final | Finalize MD5 hash |
| Crypt_SHA1_Init | Init SHA1 checksum generator |
| Crypt_SHA1_Update | Update SHA1 hash |
| Crypt_SHA1_Final | Finalize SHA1 hash |
| Crypt_AES_MakeEncryptKey | Make AES encrypt key |
| Crypt_AES_MakeDecryptKey | Make AES decrypt key |
| Crypt_AES_Encrypt | Encrypt AES message |
| Crypt_AES_Decrypt | Decrypt AES message |

## PPP Client Functions

| | |
|---|---|
| PPP_Client_Installed | Check if PPP client is installed |
| PPP_Client_Open | Open a PPP connection (IPv4) |
| PPP_Client_Open_IPv6 | Open a PPP connection with IPv6 address config |
| PPP_Client_Close | Close a PPP client connection |
| PPP_Client_GetStatus | Get PPP client status |
| PPP_Client_Get_DNSIP | Get DNS IP |
| PPP_Client_SetOptions | Set PPP options for the PPP client |
| PPP_Client_GetIP | Get IP Address of the PPP Client |

## PPP Server Functions

| | |
|---|---|
| PPP_Server_Installed | Check if PPP server is installed |
| PPP_Server_Suspend | Suspend the PPP server task |
| PPP_Server_Activate | Activate the PPP server |
| PPP_Server_GetStatus | Get the current state of the PPP server |
| PPP_Server_Get_Cfg | Get configuration data of the IPv4 PPP server |
| PPP_Server_Get_Cfg_IPv6 | Get configuration data of the IPv6 PPP server |

| | |
|---|---|
| PPP_Server_SetOptions | Set PPP options for the PPP server |
| PPP_Server_GetIP | Get IP Address of the PPP Server |

## Other Functions

| | |
|---|---|
| --- Monitoring --- | |
| Get_FTP_Login_Counters | Access the FTP server login counters |
| Get_Telnet_Login_Counters | Access the Telnet server login counters |
| Get_Telnet_State | Check if have an active telnet session |
| Get_Installed_Servers | Get information about running servers |
| Get_TCPIP_Memory_Status | Get information about TCP/IP stack memory usage |
| Get_TCPIP_Statistics | Access the internal network packet counter |
| Get_SNMP_Data | Get internal TCP/IP SNMP variables |
| Get_SNMP_DataEx | Get internal TCP/IP SNMP variables (Extended) |
| --- DHCP --- | |
| DHCP_use | Set/Reset DHCP usage of the Ethernet interface |
| Get_DHCP_Status | Get DHCP status of the Ethernet interface |
| Get_DHCP_Status_Ext | Get DHCP status of the Ethernet interface |
| --- ping --- | |
| Ping_Open | Open and start ICMP echo requests |
| Ping_Close | Finish ICMP echo requests |
| Ping_Statistics | Retrieve ping information |
| --- Server Time-outs --- | |
| Set_FTPServer_Idle_Timeout | Set FTP server time-out |
| Set_TelnetServer_Idle_Timeout | Set Telnet server time-out |
| Get_FTPServer_Idle_Timeout | Get FTP server time-out value |
| Get_TelnetServer_Idle_Timeout | Get Telnet server time-out value |
| --- Gateway --- | |
| AddDefaultGateway | **Deprecated!** Add the default gateway |
| AddDefaultGatewayEx | Add the default gateway (Extended) |
| DelDefaultGateway | Delete the default gateway |
| GetDefaultGateway | Get the current default gateway |
| AddStaticRoute | Add a route for an interface |
| AddStaticRouteEx | Add a route for an interface (extended) |
| DelStaticRoute | Delete a route from an interface |
| --- ARP cache access --- | |
| DelArpEntryByPhys | Delete ARP entry by physical address |
| AddArpEntry | Add an entry in the ARP table |
| GetARPRouteCacheEntries | Read ARP cache table |
| --- IP Multicast --- | |
| Add_IGMP_Membership | Install an IP multicast address entry |
| Drop_IGMP_Membership | Delete an IP multicast address entry |

| | |
|---|---|
| IPMulticast_to_MacAddr | Map IP multicast address to Ethernet address |
| --- TCP/IP Device driver --- | |
| Dev_Open_Interface | Install user device driver |
| Dev_Close_Interface | Close user device driver/interface |
| Dev_Recv_Interface | Receive and process incoming data |
| Dev_Recv_Wait | Waits for data received signal |
| Dev_Notify_ISR | Signal from Interrupt Service Routine |
| Dev_Get_Buffer | Get a buffer from TCP/IP stack |
| Dev_Send_Complete | Signal message send complete |
| Dev_Wait_DHCP_Complete | Waits for DHCP completion |
| Dev_PPP_Register_Chap | Register a PPP CHAP authentication callback function |
| Dev_PPP_Register_Pap | Register a PPP PAP authentication callback function |
| Dev_PPP_Set_Option | Set PPP options |
| Dev_Get_IP | Get IP address of a user defined device |
| Dev_PPP_Get_Peer_IP | Get IP address of a PPP peer |
| Dev_PPP_Get_Peer_IPv6 | Get IPv6 address of a PPP peer |
| Dev_PPP_Get_Peer_DNSIP | Get IP address of the DNS server from a PPP peer |
| Dev_Get_Handle_By_Name | Get device driver handle by device name |
| Dev_Config_Iface | IPv4 device configuration |
| Dev_Config_Iface_IPv6 | IPv6 configuration of an interface |
| Dev_UnConfig_Iface | Remove IPv4 or IPv6 device configuration from particular home index |
| --- Callback Installers --- | |
| RegisterCallbackFunction | Register a user TCP callback function |
| RegisterIPCallbackHandler | Register an User callback handler for incoming IP packets |
| RegisterIPCallbackHandler_Snd | Register an User callback handler for outgoing IP packets |
| Register_ARP_User_CB | Register an ARP User callback handler. |
| --- Miscellaneous --- | |
| api_sleep | **Deprecated!** Sleep for specified number of milliseconds |
| Reconfigure_ethernet | Reconfigure Ethernet interface |
| Get_Iface_Entries | **Deprecated!** Read table of TCP/IP device interfaces |
| Get_Iface_Entries_Ext | Read table of TCP/IP device interfaces with all IPv4 configurations/per interface |
| SetIfaceMTU | Set MTU of an device interface |
| Get_IPv6_Iface_Entries | TCP/IP IPv6 device interface information |

List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---

## helper_alloc_rtos_fast_mem

Allocate memory from @CHIP-RTOS fast memory pool (BIOSIntA0h function 57h)

```
unsigned int helper_alloc_rtos_fast_mem(unsigned long size);
```

## Parameters

*size*

Requested memory size in bytes

## Return Value

If 0, memory allocation failed. Else segment of the allocated memory area

## Comments

The returned segment can be used to build a normalized pointer to the allocated memory area in the manner shown here:

```
seg=helper_alloc_rtos_fast_mem(1024);
ptr = MK_FP(seg,0);
```

This allocated memory can be released with the **helper_release_rtos_mem** API.

**Note:**

The software Cryptographic Engine used by **SSH**, **SSL** or **IPsec** will operate noticeably quicker during key negotiation (e.g. Diffie-Hellmann, RSA, DSS) if a block of memory in the fast memory pool is available. A memory block of roughly 25 Kbyte size is required by the Cryptographic Engine. If the block is not available in this fast RAM area, then an attempt is made to allocate it from the normal system memory as a last resort.

## See Also

**helper_alloc_rtos_mem** **helper_release_rtos_mem** **helper_assign_mem**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS - User specific TCP/IP Device driver



## Introduction

**Adding a user specific device driver/linklayer interface to the TCP/IP stack**

The @CHIP-RTOS of the IPC@CHIP® provides four internal TCP/IP device interfaces:

1. Ethernet controller
2. PPP server
3. PPP client
4. Internal loopback (Virtual loopback device with IP address 127.0.0.1)

For each of these internal devices the necessary specific driver functions are implemented inside of the @CHIP-RTOS.

The provided **TCP/IP API calls 0x9F - 0xB1** allow the application developer to install an additional TCP/IP driver interface for a connected hardware device (e.g. an additional UART or an Ethernet controller). This new interface then has its own IP configuration and is used by the TCP/IP stack for IP communication in the same way as with the pre-installed internal devices.

The following sections explain how to implement and install a user specific device driver for TCP/IP. The generic example code shown here uses C-library functions provided for the TCP/IP API, which can be found in source file `TCPIP.C`. We are also using several functions from the Beck C-library. This C-library with source files is available at **www.beck-ipc.com** in the Internet download area of the IPC@CHIP®. All needed TCP/IP related types and constants are declared in the C-library header files `TCPIPAPI.H` and `TCPIP.H`.

**Important notes:**

1. IP configuration of the user device is not adjustable with the settings in the **IP section** of chip.ini configuration file. You can create your own section in chip.ini for storing IP configuration of the new device interface with **BIOSINT 0xA0 functions 0x23/0x24**.

2. Setting the default gateway (reachable via the installed interface) is now possible with the expanded **ADD_DEFAULT_GATEWAY** API.

In the following sections we describe the set of driver specific functions you may have to provide. Possible implementations of an interrupt service function and receiver task for the device interface are provided. These device driver functions must be installed with the API call **DEVOPENIFACE**. These are callback functions which are invoked by the internal TCP/IP stack of the @CHIP-RTOS. ***Do not call these functions directly from within your application!***

The driver functions are internally locked by semaphores and block every other device driver function. Because of this behavior, it's not advisable to wait (sleep) for long periods of time as this can lead to deadlock situations (primarily between the send and recv calls).

- **Device Open function**
- **Device Close function**
- **Device Send function**
- **Device Receive function**

- Device FreeReceive function
- Device IO Control function
- Device Get PhysicalAddress function
- Implementation of an interrupt service routine (ISR) and receiver task
- Install the device driver

---

## Device Open function

The TCP/IP stack calls this (optional) function to initialize the hardware and (optional) to install an Interrupt Service handler. If Borland C or Paradigm Beck IPC Edition compilers are used the driver functions must be declared as `huge` (see below). Microsoft C users must declare driver functions as `far _saveregs _loadds` .

The function should return 0 if initialization was successful. If initialization failed, return -1.

Generic Example:

```
int huge myDevOpen(DevUserIfaceHandle ifaceHandle)
{
    // Install (if necessary) a RTOS Interrupt Service function with
    //  HWAPI handler function 0xA1 service 0x84
    return 0;
}
```

Top of list
Index page

---

## Device Close function

The TCP/IP stack will execute this (optional) function when the device driver interface is closed with the DEV_CLOSE_IFACE API call.

This callback function should return 0 if the closing of the device was successful, otherwise -1 on failure.

Generic example:

```
int huge myDevClose(DevUserIfaceHandle ifaceHandle)
{
   // DeInitialize the hardware
   // Remove the ISR handler
   return 0;
}
```

Top of list
Index page

---

## Device Send function

This callback function is used by the TCP/IP stack to send the data out the device. The TCP/IP stack does not call this function from within a separate transmit task. This callback executes in the thread which made the send call, e.g. API_SEND API.

This callback function should return 0 if the sending of data was successful, otherwise -1 on error.

**Important:**

If the input parameter `flag`has value 1 (this indicates this is last frame in block) you must call

**DEV_SND_COMPLETE** to tell the TCP/IP stack that the send buffer is no longer in use.

Generic example:

```
int huge myDevSend(DevUserIfaceHandle ifaceHandle,
                   unsigned char far * dataPtr,
                   int dataLength,
                   int flag )
 {
     // Hardware specific: Send the data (dataPtr) out the device

     // Do not wait(sleep) here for indefinite times,
     //  to avoid blocking of other function calls.

     // Is this now the last frame in message block?
     if (flag & 0x1)    // Bit0 flag set?
     {
         // Inform TCP/IP stack that transmit buffer is now free.
         Dev_Send_Complete(ifaceHandle);  // C-Lib
     }
     return 0;
 }
```

## Related Topics

**DEV_SND_COMPLETE** API - Dev_Send_Complete's implementation
**DevUserIfaceHandle** type definition

**Top of list**
**Index page**

---

## Device Receive function

In this function, a received packet is passed back up into the protocol stack. The TCP/IP stack calls this function to receive a data frame from the device. TCP/IP calls this function from within your separate receiver task, which you are required to create (see final **example**).

The receive callback should return 0 if receiving of data was successful, otherwise -1 on failure.

**Important** :

It is optional but recommended to store incoming data in a buffer from the TCP/IP pre-allocated memory pool (see **TCPIPMEM**). API call **DEV_GET_BUF** returns you a buffer pointer for storing the incoming data (see example below).

If you are using your own buffer allocation for storing the incoming data, you must null out the location referenced by the input parameter `bufferHandle` (see example). In this case, you should also implement and install the device driver **callback** function:

```
int (far * DevFreeRecvFunc)(DevUserIfaceHandle ifaceHandle,
                    unsigned char far * dataPtr);
```

The TCP/IP stack calls this function to indicate that the receive buffer is no longer used by TCP/IP. The vector to this callback is placed in the `DevFreeRecv` member of the **DevUserDriver** structure at the **DEV_OPEN_IFACE** call.

Two generic examples for receiver functions follow:

myDevReceive1 : Using buffer from the TCP/IP memory pool
myDevReceive2 : Using your own receive buffer

```
// Generic example using TCP/IP memory pool receive buffer:

int huge myDevReceive1(DevUserIfaceHandle ifaceHandle,
                       unsigned char far * far * dataPtr,
                       int far * dataLength,
                       DevUserBufferHandle bufferHandle)
{
    int                 errorCode;
    unsigned int        rcvdLength;
    unsigned char far  *tcp_buffer ;

    // Hardware specific: Check how many incoming data bytes are available.
    rcvdLength = .....;   // =byte count

    // Get a buffer from TCP/IP by calling API service 0xA5 and save at dataPtr
    // This API call also sets the bufferHandle for internal use by TCP/IP stack.
    Dev_Get_Buffer(bufferHandle, dataPtr, rcvdLength);  // C-Lib call
    tcp_buffer = *dataPtr ;       // Check if memory allocation successful
    if (tcp_buffer != (unsigned char far *)0)
    {
       // Hardware specific: Move received data from device to tcp_buffer
       // Do not wait (sleep) here for indefinite times,
       //   to avoid blocking of other function calls.

       *dataLength = recvdLength;   // Report number of bytes now in tcp_buffer
       return 0;  // success
    }
    else
    {
       return -1; // out of memory
    }
}


 // Generic example for using your own receive buffer:

int huge myDevReceive2(DevUserIfaceHandle ifaceHandle,
                       unsigned char far * far * dataPtr,
                       int far * dataLength,
                       DevUserBufferHandle bufferHandle)
{
    // Save the pointer to the beginning of the data
    *dataPtr = myBuffer;  // myBuffer somehow allocated by the user

    // Hardware specific: Read data from your device and store in myBuffer

    // Save the length (in bytes) of received data
    *dataLength = deviceDataLength;

    // IMPORTANT: Null out the bufferhandle pointer
    *bufferHandle = (DevUserBuffer)0;
    return 0;
}
```

## Related Topics

**DEV_GET_BUF** API - Dev_Get_Buffer's implementation
**DevUserIfaceHandle** type definition

**Top of list**
**Index page**

---

# Device FreeReceive function

Implementation of this function is necessary if you decide to use your own buffers for receiving incoming data.

The TCP/IP stack will call this function to inform you that the receive buffer (input parameter `dataPtr`) is no longer used by TCP/IP.

This callback should return 0 if ok, else -1 on failure.

Generic example:

```
int huge myDevFreeRecv(DevUserIfaceHandle ifaceHandle, unsigned char far *dataPtr )
{
     // Somehow free your allocated buffer at dataPtr
     my_free(dataPtr);

     return 0;
}
```

### Related Topics

**DevUserIfaceHandle** type definition

**Top of list**
**Index page**

---

## Device IO Control function

This function allows to react on IO control requests of the TCP IP stack. The IO Control function will be called when the IP configuration of the specific interface was modified.

This callback should return 0 if ok, else -1 on failure.

Generic example:

```
int huge myDevIoctl(DevUserIfaceHandle ifaceHandle,
                    int        flag,
                    void far * optionPtr,
                    int        optionLen )
{
     char ipStr[50];
     char nameStr[20];

     if (flag == DEV_IOCTL_IPCFG_NOTIFY)
     {
       // do update ip address in the CHIP.INI file
       DevIpv4IfaceCfg   *pDevIpv4IfaceCfg = optionPtr;
       if (pDevIpv4IfaceCfg->IpAddr)
       {
          // setup ip address
          InetToAscii(&pDevIpv4IfaceCfg->IpAddr, ipStr);
          snprintf(nameStr, sizeof(nameStr), "ADDRESS%01u", pDevIpv4IfaceCfg->multiHomeIndex);
          BIOS_Set_Ini_String("IP_MYITF", nameStr, ipStr );

          // setup netmask
          InetToAscii(&pDevIpv4IfaceCfg->Netmask, ipStr);
          snprintf(nameStr, sizeof(nameStr), "NETMASK%01u", pDevIpv4IfaceCfg->multiHomeIndex);
          BIOS_Set_Ini_String("IP_MYITF", nameStr, ipStr );

          // disable DHCP, when mulit home index 0 becomes static
          if (pDevIpv4IfaceCfg->multiHomeIndex == 0)
          {
            BIOS_Set_Ini_String("IP_MYITF", "DHCP", "0" );
          }
       }
       else
       {
          // DHCP is allowed for multi home index 0 only
          if (pDevIpv4IfaceCfg->multiHomeIndex != 0) return 0;

          // enable DHCP
          BIOS_Set_Ini_String("IP_MYITF", "DHCP", "1" );
       }
```

```
    }

    if (flag == DEV_IOCTL_IPV6CFG_NOTIFY)
    {
       DevIpv6IfaceCfg   *pDevIpv6IfaceCfg = optionPtr;

       // do update IPv6 address (analog to procedure above)
       // ...
    }

    return 0;
}
```

### Related Topics

**DevIpv4IfaceCfg** type definition
**DevIpv6IfaceCfg** type definition

**Top of list**
**Index page**

---

## Device Get PhysicalAddress function

This function applies only to Ethernet controllers.

The 6 byte array referenced by the `PhysicalAddress` input parameter should be filled with the MAC address of your connected Ethernet controller.

Generic example:

```
int huge myDevGetPhysAddr(DevUserIfaceHandle ifaceHandle,
                          unsigned char far * physicalAddress)
{
    // Hardware specific: copy MAC address into physicalAddress
    _fmemcpy(physicalAddress, myEthernet_MAC, 6) ;
    return 0;
}
```

### Related Topics

**DevUserIfaceHandle** type definition

**Top of list**
**Index page**

---

## Implementation of an interrupt service routine (ISR) and receiver task

The implementation of a device specific ISR is optional. If your hardware device is able to generate interrupts on device events (e.g. incoming data available), you can implement an ISR like the example below. The CPU time spent within an ISR must be keep to a minimum, as the length of this interrupts masked period impacts the interrupt latency of the other critical system ISR's. Consequently, your ISR should only notify events (incoming data, error,..) at the device and not directly handle device events itself (e.g. retrieve incoming data) immediately within the ISR.

With API call **DEV_NOTIFY_ISR** the ISR should wakeup a user provided task, which receives the incoming data from the device and moves the data into the TCP/IP stack. This task should use API call **DEV_RECV_WAIT** and **DEV_RECV_IFACE** (see example below).

Instead of a creating a new task, it is also possible to use your program's main thread for receiving by having it perform the `MyReceiveTask()` actions shown below.

If your device doesn't support interrupts, you could create a polling task (or again, simply use your program's main thread for this purpose) which periodically checks your device for incoming data as illustrated in the `MyReceiveTask_Polling` example below.

**Important** : An ISR must be installed as a RTOS ISR with the **Install Interrupt Service Routine** of the Hardware API.

Generic examples for an ISR and a two forms of receiver task functions follow.

```
// Interrupt Service Routine

void interrupt MyDeviceISRHandler(void)
{
   int receivedFrames;
   int errorCode;

   // Hardware specific: Check if there are incoming data packets available

   // Wakeup receiver task
   Dev_Notify_ISR(MyDevHandle, receivedFrames, 0, &errorCode);  // C-Lib call

   // Note: Issue no EOI here.
   //   (EOI for the ISR is issued inside of the CHIP-RTOS.)
}

 // Generic example for a receiver task function, which waits for an event from ISR:

void huge MyReceiveTask(void)
{
    int errorCode;
    int statRecv;
    // Optional: do some initialization

    while(1)
    {
       // Wait for a wakeup from ISR
       Dev_Recv_Wait(mydevdriver.IfaceHandle, &errorCode);   // C-Lib call

       // After wakeup received and move incoming data into the stack
       do
       {              // C-Lib call
          statRecv = Dev_Recv_Interface(mydevdriver.IfaceHandle, &errorCode);
       } while (statRecv != -1);
    }
}


 // Generic example for receiver task, polling for incoming data:

void huge MyReceiveTask_Polling(void)
{
    int errorCode;

    // Optional: do some initialization

    // Wait for completion of interface installation (Intr 0xAC 0xA0)
    while (install_done == 0)
    {
       RTX_Sleep_Time(10); // Go to sleep for a defined time.
    }

    while(1)
    {
       // Check if there is data available inside of your device.
       if (myDeviceDataAvail())
       {
           // Receive and move incoming data into the stack
           Dev_Recv_Interface(mydevdriver.IfaceHandle, &errorCode);
       }
       RTX_Sleep_Time(10); // Go to sleep for a defined time.
    }
}
```

### Related Topics

**DEV_NOTIFY_ISR** API - Dev_Notify_ISR's implementation
**DEV_RECV_WAIT** API - Dev_Recv_Wait's implementation
**DEV_RECV_IFACE** API - Dev_Recv_Interface's implementation
**RTX_SLEEP_TIME** API - RTX_Sleep_Time's implementation

**Top of list**
**Index page**

---

## Install the device driver

Based on the previous sections of this document, the following generic example should make clear the main steps required to install a user implemented device driver:

```
#include "tcpip.h"
#include "rtos.h"

int huge myDevOpen(DevUserIfaceHandle ifaceHandle);

int huge myDevClose(DevUserIfaceHandle ifaceHandle);

int huge myDevSend(DevUserIfaceHandle ifaceHandle,
                   unsigned char far * dataPtr,
                   int dataLength, int flag);

int huge myDevReceive1(DevUserIfaceHandle ifaceHandle,
                       unsigned char far * far * dataPtr,
                       int far * dataLength,
                       DevUserBufferHandle bufferHandle);

int huge myDevGetPhysAddr(DevUserIfaceHandle ifaceHandle,
                          unsigned char far * physicalAddress);

void interrupt MyDeviceISRHandler(void);

void huge MyReceiveTask(void);

unsigned int       recvID;              // task ID
unsigned int       myrecv_stack[1024];  // stack for receiver task
unsigned char      install_done = 0;    // waiting flag  for receiver task
unsigned int       errorCode;
DevUserIfaceHandle MyDevHandle;

TaskDefBlock  myrecv_defblock =
{
    MyReceiveTask,                 // task function
    {'D','E','V',' '},             // a name: 4 chars
    &myrecv_stack[1024],           // top of stack
    1024*sizeof(int),              // size of stack
    0,                             // attributes, not supported
    20,                            // priority 20(high) ... 127(low)
    0,                             // no time slicing
    0,0,0,0                        // mailbox depth,
};

char far * mydevicename  = "MyDev";
char far * IPString      = "192.168.200.020";
char far * NetmaskString = "255.255.255.000";

DevUserDriver mydevdriver;

int main(void)
{
    //*********************************************************************
    // Initialize struct mydevdriver;
    //*********************************************************************
    mydevdriver.DevName = mydevicename;    // Unique device name,
```

```
    //  max. 13 chars + 0.

    inet_addr(IPString     , &mydevdriver.IpAddr);  // IP address
    inet_addr(NetmaskString, &mydevdriver.Netmask); // Netmask

    mydevdriver.iface_type = DEV_ETHERNET;          // Ethernet device
    mydevdriver.use_dhcp  =  0;                     // no DHCP

    //Important:
    // At the first DEV_OPEN_IFACE call for a device, IfaceHandle must be NULL.
    mydevdriver.IfaceHandle = 0 ;

    // Note: If the interface should be restarted by calling DEV_CLOSE_IFACE
    //  and DEV_OPEN_IFACE (e.g. for changing IP configuration) the
    //  IfaceHandle must contain at DEV_OPEN_IFACE the valid IfaceHandle handle
    //  from the first DEV_OPEN_IFACE call.

    // Install your driver functions
    mydevdriver.DevOpen        = (void far *)mydevOpen;
    mydevdriver.DevOpen        = (void far *)mydevClose;
    mydevdriver.DevSend        = (void far *)mydevSend;
    mydevdriver.DevRecv        = (void far *)myDevReceive1;
    mydevdriver.DevFreeRecv    = (void far *)0;  // Since using TCP/IP buffers
    mydevdriver.DevGetPhysAddr = (void far *)myDevGetPhysAddr;
    mydevdriver.DevIoctl       = (void far *)myDevIoctl;

    //*************************************************************************
    // Install the device driver interface
    //*************************************************************************
    result = Dev_Open_Interface(&mydevdriver, &errorCode);  // C-Lib

    // if(result).....

    //*************************************************************************
    // Create the receiver task
    //*************************************************************************
    result = RTX_Create_Task(&recvID, &myrecv_defblock);   // C-Lib

    // if(result).....

    // Optional, but recommended: Change receiver task to a high priority.
    RTX_Change_TaskPrio(recvID, 4, &errorCode);

    //*************************************************************************
    //If device interface should be configured by DHCP, wait for completion
    //of the DHCP configuration process
    //*************************************************************************
    if (mydevdriver.use_dhcp == 1)
    {
        result= Dev_Wait_DHCP_Complete(&mydevdriver, 20, &errorCode); // C-Lib
        // if(result)....
    }
    //  Wait forever, or stay resident with int21h call 31h.
    //  It is possible to close and restart the interface inside of an application.
    //  But due to the internal architecture of the TCP/IP stack, it is not
    //  possible to exit the driver program and restart the interface with the
    //  same unique name.
    //  Your driver program should run forever.  Avoid killing the receiver task.

    while(1) RTX_Sleep_Time(100);

}// End of main(void)
```

## Related Topics

**API_INETADDR** API - inet_addr's implementation
**DEV_OPEN_IFACE** API - Dev_Open_Interface's implementation
**DEV_WAIT_DHCP_COMPLETE** API - Dev_Wait_DHCP_Complete's implementation
**RTX_TASK_CREATE** API - RTX_Create_Task's implementation
**RTX_SLEEP_TIME** API - RTX_Sleep_Time's implementation
**DevUserDriver** data structure type definition

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

End of document

## RTX_FineTickInit

Initialize data required by the **RTX_GetFineTick** and **RTX_GetTick_us** API.

```
void RTX_FineTickInit (void);
```

## Return Value

-- none --

## Comments

This function is implicitly called by the **RTX_GetFineTick** and **RTX_GetTick_us** API if it has not already been called explicitly by the user.

Interrupts are enabled within the @Chip-RTOS API call made here. Consequently, it may be advisable to call this function from your application program's initialization area to avoid this action during your real-time operation. After a single call here, this one-shot function performs no operation on subsequent calls since its job is already done.

Some global constants are initialized by this API. This data is used internally by the other C-library functions, but may also be of general use to the application. (The names given these have been preceded with an underscore to avoid name space conflicts.) These constants are listed here.

**unsigned long far *_pRti_Tick ;**

This pointer references the 32 bit Real-Time Interrupt up counter internal to the @Chip-RTOS. ***This location should never be written to from outside the @Chip-RTOS!*** Note that for reliable reads, interrupts must be momentarily masked if both 16 bit halves are sampled.

**unsigned int _RTI_kHz ;**

The Real-Time Interrupt **rate**in kHz. (For SC1x systems this value will always be 1.)

**unsigned int _us_per_RTI ;**

Number of microseconds per Real-Time Interrupt. (For SC1x systems this value will always be 1000.)

**unsigned int _Timer2_Limit ;**

Hardware TIMER2 ticks per Real-Time Interrupt.

**unsigned int _Timer2Tick_per_us ;**

Number of hardware TIMER2 ticks per microsecond.

## SC1x3/SC23 Comments

For some selected Real-Time Interrupt rates, the `_us_per_RTI` and `_Timer2Tick_per_us` values computed here will not be accurate. This is the case for any **RTI** rates which do not divide 1000 an integral number of times, e.g. 15 kHz. Consequently the **RTX_GetTick_us** API should not be used for such configurations, since the computation done within that API relies on the accuracy of these constants.

## See Also

**RTX_GetFineTick** **RTX_GetTick_us**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Batch_Continue

Continue **batch file** execution.

```
void BIOS_Batch_Continue ( void );
```

## Parameters

-- none --

## Return Value

-- none --

## Comments

This call allows the next program listed in a batch file to start execution. This is necessary when the **BATCHMODE** is set to '1' for sequential execution of programs.

This function is implemented by waking up a batch file execution task which dispatches any subsequent program listed in the batch file.

## See Also

**BIOS_Set_Batch_Mode**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

# @CHIP-RTOS - TCP/IP Error Codes

## TCP/IP API Error Codes

Network API error codes returned by API calls (stated here in decimal):

IPsec error codes

- PPPclient
- PPPstatus
- DNSdns
- PKIerror

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
224 No resource available
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 serial port or PPP server already uses port.
        // This error code also occurs, if
        // the PPP client is interrupted while dialing
        // (e.g. user break by setting the flag modem_break
        //  in the  pppclient_init structure
        //  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  // Opening 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    // Link opening in progress
#define PPP_LNKUP        2    // Link is established
```

**Top of list**
**Index page**

---

## DNS resolver error codes

```
#define DNS_ERROR_ILLEGAL_ARGUMENT     -1   // An argument with an invalid value was passed
#define DNS_ERROR_RESOURCE_LACK        -2   // Lack of a resource such as memory or sockets
#define DNS_ERROR_SERVERS_UNREACHABLE  -3   // No name server could be reached
#define DNS_ERROR_NO_RESOLUTION        -4   // No resolution could be found
```

## PKI error codes

```
// Possible errors of PKI certificate add function
#define PKI_CERT_BADCERT        0x01  // Critical error
#define PKI_CERT_REVOKED        0x02  // Revoked certiticate
#define PKI_CERT_EXPIRED        0x04  // Expired certificate
#define PKI_CERT_UNMATCHED      0x08  // Identity not matched
#define PKI_CERT_UNKNOWNCA      0x10  // CA is unknown
```

Top of list
Index page
TCP/IP API Listing

End of document




# @CHIP-RTOS - TCP/IP Application Developers Note

IPC@CHIP® Documentation **Index**

## TCP/IP Applications

### Developer Notes

For better understanding of programming TCPIP applications Beck offers several examples with source code. The given examples may be used and modified by the application programmer. The application programmer should know about how the socket-interface works and reading these examples can help with gaining that understanding.
Examples for e.g. programming UDP/TCP echo client/server applications and much more can be found at **www.beck-ipc.com**.

All example programs use the Beck C-library API to access the @CHIP-RTOS functions. It is important that the current version of the Beck C-library is used to match the current RTOS version.

TCP/IP **API**

End of document




# @CHIP-RTOS - Programming client server applications

IPC@CHIP® Documentation **Index**

## Client-Server

Here is a short description for programming client/server applications with the TCP/IP API.

The most often 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 (client).

An example for the simplest service is the standard echo server 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.

A TCP/IP stack provides two different methods for client-server connections, UDP and TCP.

1) **UDP protocol:**

This protocol realizes connectionless communication between a client and server, based on sending and receiving of single datagrams.

TCP/IP API calls used by a 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 the socket**

TCP/IP API calls used by a 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 using a prerecorded endpoint address.**

2) **TCP protocol:**

The TCP protocol is a connection- and byte stream-oriented protocol.

TCP/IP API calls used by a TCP client:

- **Open a socket**

- **Connect to a remote peer**
- **Send an outgoing stream of characters**
- **Receive an outgoing stream of characters**
- **Close the socket**

TCP/IP API calls used by 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**

Examples with source code for programming UDP and TCP echo client/server applications can be found at **www.beck-ipc.com**.

Note:
All program examples are built with Beck C-Library functions declared in the library header files TCPIP.H and TCPIPAPI.H.

---

End of document

---

## select

Await specified events on a selected set of sockets.

```
int select ( int nfds,
             fd_set far *readfds,
             fd_set far *writefds,
             fd_set far *exceptfds,
             const struct timeval FAR *timeout ) ;
```

## Parameters

*nfds*

One greater than highest socket descriptor to be monitored in the following set of three `fd_set` objects.

*readfds*

Input/Output Parameter: Pointer to a **fd_set** bit field that on input specifies the set of sockets to be monitored for receiver activity. On return those sockets for which receiver data is ready among those specified for monitoring will have their bits in this field set to '1'. All other bits will be zeroed.

*writefds*

Input/Output Parameter: Pointer to a **fd_set** bit field that on input specifies the set of sockets to be monitored for send ready condition. On return those sockets ready for send among those specified for monitoring here will have their bits in this field set to '1'. All other bits will be zeroed.

*exceptfds*

Input/Output Parameter: Pointer to a **fd_set** bit field that on input specifies the set of sockets to be monitored for error conditions. On return those sockets with errors among those specified for monitoring here will have their bits in this field set to '1'. All other bits will be zeroed.

*timeout*

Input Parameter: Pointer to a **timeval** data structure which specifies the maximum wait time. This pointer can be NULL to block until an event occurs on one of the specified sockets.

## Return Value

0 if timeout, otherwise number of hits. This is the number of selected sockets for which a specified event has occurred.

If `nfds` input parameter was invalid, then the return value will also be zero.

## Comments

This function and its **select_ms()** variant API operate similarly to the standard Berkeley sockets implementation. This API is useful when a single program thread is to manage the activity on more than one socket. The thread can sleep for up to a specified amount of time, awaiting events on a selected set of sockets. These events, selected by the caller, can be receiver activity, transmitter activity and/or socket error conditions.

The most common use of this API is to sleep on a combination of receiver activity or error conditions. In this manner, your thread will wakeup when either data is received by one of your specified sockets, or if one of the sockets is closed by the peer. For this usage, the `writefds` parameter would be NULL to indicate that you are not interested in hearing about any socket transmit activity.

Note that when only a single socket is involved, the **recv()** and **recvfrom()** API provide a more efficient receiver sleep mechanism than `select()`.

This API's three pointer parameters to **fd_set** objects can point to the same object. For example, for the receiver activity wait call, the `readfds` and `exceptfds` parameters could both reference the same `fd_set` object. This is commonly done this way. You lose the resolution over whether the error or receiver activity event occurred, but this can be resolved on a subsequent **recv()** (or **recvfrom()**) call that would normally follow a return from the `select()`.

In the parameter description above for `readfds` , `writefds` and `exceptfds` it says *"All other bits will be zeroed"* . Here "all bits" means those bits with indexes lower than the `nfds` input parameter value. (Caller is responsible for zeroing the others with indexes higher). This assumes that either no two pointers are pointing to the same `fd_set` or that the other events have not occurred. The point here is that zeros are returned for sockets which have no event. For the case where a pointer to the same `fd_set` object is passed for both the `readfds` and `exceptfds` parameters, for example, the returned bits will be set if either receiver data is ready or an error condition has occurred. The bits set represent a logical OR of input parameter conditions which referenced the `fd_set`.

The `nfds` can be jammed to the `MAX_BECK_SOCKETS` constant value and this API will operate. (The lazy programmer's approach.) However, efficiency down inside the TCP/IP stack is improved if you confine the bit field range over which the system must operate by setting the `nfds` input parameter to one count greater than your highest socket descriptor bit set in any of your `fd_set` fields.

This API is reentrant so long as it is called for a mutually exclusive set of sockets. However, when two threads call concurrently with an overlapping set of sockets specified, the last thread to call will receive a zero hit count return value. This should be avoided.

The alternative non-standard calling convention offered by **select_ms()** is recommended when Berkeley sockets compatibility is not required. This other API works the same except for that it uses the simplified timeout argument specified in milliseconds.

## See Also

**select_ms** **recvfrom** **recvfromto**
**recv** **FD_Next**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## select_ms

Await specified events on a selected set of sockets.

```
int select_ms ( int nfds,
                fd_set far *readfds,
                fd_set far *writefds,
                fd_set far *exceptfds,
                long timeout ) ;
```

## Parameters

*nfds*

One greater than highest socket descriptor to be monitored in the following set of three `fd_set` objects.

*readfds*

Input/Output Parameter: Pointer to a **fd_set** bit field that on input specifies the set of sockets to be monitored for receiver activity. On return those sockets for which receiver data is ready among those specified for monitoring will have their bits in this field set to '1'. All other bits will be zeroed.

*writefds*

Input/Output Parameter: Pointer to a **fd_set** bit field that on input specifies the set of sockets to be monitored for send ready condition. On return those sockets ready for send among those specified for monitoring here will have their bits in this field set to '1'. All other bits will be zeroed.

*exceptfds*

Input/Output Parameter: Pointer to a **fd_set** bit field that on input specifies the set of sockets to be monitored for error conditions. On return those sockets with errors among those specified for monitoring here will have their bits in this field set to '1'. All other bits will be zeroed.

*timeout*

Timeout in milliseconds. Set to a minus value to block without timeout. Set to zero for polling.

## Return Value

0 if timeout, otherwise number of hits. This is the number of selected sockets for which a specified event has occurred.

If `nfds` input parameter was invalid, then the return value will also be zero.

## Comments

The alternative to **select()** API is available when Berkeley sockets compatible source is desired. The only difference is that this function takes the timeout value as a simple long value in units of milliseconds.

Refer to the **select()** API description for a more information about these API.

## See Also

**select** **FD_Next**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## getHostByName

This function can be used to find the IPv4 addresses of a remote host associated to its domain name via the Domain Name Service (DNS).

```
short getHostByName( const char far *domainName,
                     unsigned char nAddresses,
                     struct in_addr far *addresses );
```

## Parameters

*domainName*

Domain name of the host (e.g. "www.beck-ipc.com")

*nAddresses*

Maximum number of IP addresses that the function is to return

*addresses*

Pointer to an array of type **struct in_addr** where the function is to store the found IP addresses; must be large enough to hold the number of IP addresses stated in nAddresses.

## Return Value

Number of addresses actually stored in the given array or negative **error code**

## Comments

DNS queries consume a relative large part of your task's stack.

## See Also

**getHostByName2** **getHostByAddr** **getHostByAddr2** **getMailHosts** **setNameServer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **DNS_GET_HOST_BY_NAME** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## getHostByName2

This function can be used to find the IPv6 addresses of a remote host associated to its domain name via the Domain Name Service (DNS).

```
short getHostByName2( const char far *domainName,
                      unsigned char nAddresses,
                      struct in6_addr far *addresses );
```

## Parameters

*domainName*

Domain name of the host (e.g. "www.beck-ipc.com")

*nAddresses*

Maximum number of IP addresses that the function is to return

*addresses*

Pointer to an array of type **struct in6_addr** where the function is to store the found IP addresses; must be large enough to hold the number of IP addresses stated in nAddresses.

## Return Value

Number of addresses actually stored in the given array or negative **error code**

## Comments

DNS queries consume a relative large part of your task's stack.

## See Also

**getHostByName** **getHostByAddr** **getHostByAddr2**
**getMailHosts** **setNameServer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **DNS_GET_HOST_BY_NAME2** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## getHostByAddr

This function can be used to find the (canonical) domain name associated with a host by one of its IPv4 addresses via the Domain Name Service (DNS).

```
short getHostByAddr( struct in_addr far *address,
                     char far *domainName );
```

## Parameters

*address*

Pointer to the host's IP address of type **struct in_addr**.

*domainName*

Pointer to a buffer where the function is to store the domain name; A domain name can be up to 254 characters long. Adding one byte for the trailing zero byte of the string, the buffer must have a size of at least 255 bytes.

## Return Value

Zero in case of success or negative **error code**

## Comments

DNS queries consume a relative large part of your task's stack.

## See Also

**getHostByName** **getHostByName2** **getHostByAddr2**
**getMailHosts** **setNameServer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **DNS_GET_HOST_BY_ADDR** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## getHostByAddr2

This function can be used to find the (canonical) domain name associated with a host by one of its IPv6 addresses via the Domain Name Service (DNS).

```
short getHostByAddr2( struct in6_addr far *address,
                      char far *domainName );
```

## Parameters

*address*

Pointer to the host's IP address of type **struct in6_addr**.

*domainName*

Pointer to a buffer where the function is to store the domain name; A domain name can be up to 254 characters long. Adding one byte for the trailing zero byte of the string, the buffer must have a size of at least 255 bytes.

## Return Value

Zero in case of success or negative **error code**

## Comments

DNS queries consume a relative large part of your task's stack.

## See Also

**getHostByName** **getHostByName2** **getHostByAddr**
**getMailHosts** **setNameServer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **DNS_GET_HOST_BY_ADDR2** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## getMailHosts

Use this function to determine the mail host(s) of a domain, i.e. host accepting internet mail for the domain.

```
short getMailHosts( const char far *domainName,
                    unsigned char nHosts,
                    DnsMailHost far *hosts );
```

## Parameters

*domainName*

Domain name to find mail hosts for (e.g. "beck-ipc.com")

*nHosts*

Maximum number of host names that the function is to return

*hosts*

Pointer to an array of type **DnsMailHost** where domain name and preference value of the mail hosts will be stored; The array must be large enough to hold the number of entries passed in nHosts.

## Return Value

Number of mail hosts actually stored or negative **error code**

## Comments

DNS queries consume a relative large part of your task's stack.

## See Also

**getHostByName** **getHostByName2** **getHostByAddr**
**getHostByAddr2** **setNameServer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **DNS_GET_MAIL_HOSTS** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## setNameServer

This function can be used to add a prioritised name server which will be queried before all other name servers. Normally the DNS resolver determines name server addresses automatically from network devices configured via DHCP or PPP. In addition you can specify two name servers in the CHIP.INI **configuration file**. Name servers found via DHCP have the lowest priotity. Next come name servers found via PPP, then the two configured in CHIP.INI. The name server added here has the highest priority. Note that calling this function a second time overwrites the name server address added first.

```
short setNameServer( int addressFamily,
                     void far *address );
```

## Parameters

*addressFamily*

Address family, AF_INET for IPv4 or AF_INET6 for IPv6

*address*

Pointer to the IP address of the name server; should point to a structure of either type `struct in_addr` or `struct in6_addr`.

## Return Value

Zero in case of success or negative **error code**

## See Also

**getHostByName** **getHostByName2** **getHostByAddr**
**getHostByAddr2** **getMailHosts**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **DNS_SET_NAME_SERVER** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS - IP Security Application Programmer's Interface



## IP Security

IP Security API

These interfaces allow control over the IP Security operation. This set of API is available only in the SC1x3/SC23 products.

IP Security **Overview**
IP Security **Data Structures**
IP Security **Known Problems**

### IP Security Interrupt 0xAC Services

Service is selected by index in AX register, where AH part is 0xCF in each case.

### IP Security Initialization

- **0xCF00: IPSEC_ADD_POLICY**
- **0xCF01: IPSEC_CLEAR_POLICY**
- **0xCF05: IPSEC_START**
- **0xCF06: IPSEC_STATUS**
- **0xCF07: IPSEC_SET_OPTIONS**
- **0xCF0A: IPSEC_RESTORE_POLICY**

### IKE Transforms

- **0xCF08: IPSEC_IKE_PHASE1_SET**
- **0xCF09: IPSEC_IKE_PHASE1_DELETE**

### Preshared Keys

- **0xCF02: IPSEC_ADD_PRESHARED_KEY**
- **0xCF03: IPSEC_REMOVE_PRESHARED_KEY**
- **0xCF04: IPSEC_CLEAR_PRESHARED_KEYS**

## Interrupt 0xAC service 0xCF00: IPSEC_ADD_POLICY

This function is used to add IP security policies.

### Parameters

*AX*
0xCF00 (= IPSEC_ADD_POLICY)

*DS:BX*
Pointer to an array of CX **tIPSEC_POLICY_PAIR** structures.

*ES:DI*
Pointer to an array of **tIPSEC_POLICY_SELECTOR** structures.

*DX:SI*
Pointer to an array of **tIPSEC_POLICY_CONTENT** structures.

*CX*
Number of data structures in array at DS:BX.

### Return Value

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

### Comments

The IP security must be enabled prior to using this API.

The **tIPSEC_POLICY_PAIR** structures at DS:BX are used to connect the members of the other two structure arrays, mapping policy selectors to policy content. Care must be taken to assure that these indexes map to existing structures within these two arrays.

Refer to the C-Library function **IPsec_Set_Policy()** for more information about this API.

### Beck C-Library Usage

**IPsec_Set_Policy()** - Set policy

### Related Topics

Clear Policy **IPSEC_CLEAR_POLICY**
Restore Policy **IPSEC_RESTORE_POLICY** from file
Set IKE transform **IPSEC_IKE_PHASE1_SET**

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xCF01: IPSEC_CLEAR_POLICY

This function deletes the IP security policies.

### Parameters

*AX*
0xCF01 (= IPSEC_CLEAR_POLICY)

### Return Value

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

### Comments

The IP security must be enabled prior to using this API.

A default policy of BYPASS for all IP addresses will be used following this action.

### Beck C-Library Usage

**IPsec_Clear_Policy()** - Clear policy

### Related Topics

Set Policy **IPSEC_ADD_POLICY**

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xCF02: IPSEC_ADD_PRESHARED_KEY

This add preshared key for use with Internet Key Exchange (IKE) protoocol with the specified remote IP address.

### Parameters

*AX*
0xCF02 (= IPSEC_ADD_PRESHARED_KEY)

*ES:BX*
Pointer to a zero terminated ASCII string specifying the remote IP address.

*DX:SI*
Pointer to zero terminated ASCII string containing the preshared key.

## Return Value

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

## Comments

The IP security must be enabled prior to using this API.

The remote IP address may be either IPv4 of IPv6. Domain names must be **resolved** first by the caller and are not supported by this API.

## Beck C-Library Usage

**IPsec_Add_Preshared_Key()**

## Related Topics

**IPSEC_REMOVE_PRESHARED_KEY**
**IPSEC_CLEAR_PRESHARED_KEYS**
CHIP.INI **PRESHARED_KEY**
**DNS_GET_HOST_BY_NAME**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xCF03: IPSEC_REMOVE_PRESHARED_KEY

This function removes a preshared key from the database for the specified remote IP address.

## Parameters

*AX*
0xCF03 (= IPSEC_REMOVE_PRESHARED_KEY)

*ES:BX*
Pointer to a zero terminated ASCII string specifying the remote IP address.

## Return Value

DX = 0, AX = 0 success

DX != 0 AX: contains error code

## Comments

The IP security must be enabled prior to using this API.

The remote IP address may be either IPv4 of IPv6. Domain names must be resolved first by the caller and are not supported by this API.

## Beck C-Library Usage

IPsec_Remove_Preshared_Key()

## Related Topics

IPSEC_ADD_PRESHARED_KEY
IPSEC_CLEAR_PRESHARED_KEYS
DNS_GET_HOST_BY_NAME

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xCF04: IPSEC_CLEAR_PRESHARED_KEYS

This function clears all preshared keys from the database used by the Internet Key Exchange (IKE) protoocol.

## Parameters

*AX*
0xCF04 (= IPSEC_CLEAR_PRESHARED_KEYS)

*-- none --*

## Return Value

DX = 0, AX = 0

## Beck C-Library Usage

IPsec_Clear_Preshared_Keys()

## Related Topics

IPSEC_ADD_PRESHARED_KEY

IPSEC_REMOVE_PRESHARED_KEY

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAC service 0xCF05: IPSEC_START

This function activates the IP Security functionality in the TCP/IP stack.

## Parameters

*AX*
0xCF05 (= IPSEC_START)

*-- none --*

## Return Value

DX = 0, AX = 0 on success
else on failure AX viewed as a signed value:

AX less than 5000: look **here**for error code description. A error code of 235, "Operation would block", is returned if IP security initialization is pending on **DHCP**IP address configuration.

5000 <= AX <6000: The **POLICY_FILE**restore operation has failed. Subtract 5100 from this value and check **here**for resulting error value.

6000 <= AX <7000: The **PRESHARED_KEY**loading failed. Subtract 6100 from this error code and look **here**for a description of the resulting error code.

## Comments

Alternatively, the IP Security can be started by setting the IPSEC **ENABLE** value in the CHIP.INI. If the function has not already been activated, then this API will respond to the **POLICY_FILE** and **PRESHARED_KEY** entries in the CHIP.INI file.

## Beck C-Library Usage

IPsec_Start()

## Related Topics

**IKE** command
**IPSEC_ADD_POLICY**
**IPSEC_RESTORE_POLICY**
**IPSEC_ADD_PRESHARED_KEY**
**IPSEC_STATUS**

Set IKE transform **IPSEC_IKE_PHASE1_SET**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xCF06: IPSEC_STATUS

Read out the status of the IP Security function.

## Parameters

*AX*
0xCF06 (= IPSEC_STATUS)

*-- none --*

## Return Value

DX = 0, AX = enumerator as follow:
IPSEC_OFF (0) - Not **started**yet.
IPSEC_FAILED (1) - Startup failed, perhaps insufficient **memory**.
IPSEC_AWAITING_DHCP (2) - Awaiting Ethernet IP address for IKE.
IPSEC_READY (3) - Ready for operation.
BX is the **error** code which is non-zero if state is `IPSEC_FAILED`.
CX is the latest IKE **error** status.

## Beck C-Library Usage

**IPsec_Status()**

## Related Topics

**IPSEC_START**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xCF07: IPSEC_SET_OPTIONS

Set IP security and IKE options.

## Parameters

*AX*
0xCF07 (= IPSEC_SET_OPTIONS)

*BX*
Option selector

*CX*
Option value

## Return Value

DX = 0, AX = 0: Success
DX = -1, AX = -1: Failure, invalid option or value

## Comments

This function allows the IP Security and Internet Key Exchange (IKE) protocol options to be adjusted at run-time. The IP Security function must have been started either from the **CHIP.INI** or **IPsec_Start()** API before using this API.

A specification of the available options and their values is provided with the **IPsec_Set_Option()** C-library function description.

## Beck C-Library Usage

**IPsec_Set_Option()**

## Related Topics

**IKE_PH1_AGGRESSIVE** definition in CHIP.INI file
**IKE_PH2_PFS** definition in CHIP.INI file
**IPSEC_START**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xCF08: IPSEC_IKE_PHASE1_SET

Specify a transform for IKE's phase 1 proposal.

## Parameters

*AX*

0xCF08 (= IPSEC_IKE_PHASE1_SET)

*ES:BX*

Pointer to a **tIPSEC_TKE_PHASE1** data structure used to specify the transform's parameters. This data is not altered by this API. A copy of this data is made, so the caller's image need not persist beyond the return from this function.

*CX*

Index 0 or 1 selects which of the two phase 1 IKE transforms is to be set.

## Return Value

DX = 0, AX = **error code**,

## Comments

This function adds a user specified phase 1 transform to IKE's phase 1 proposal. The user is able to configure up to two such transforms. The most preferred has index 0, the less preferred has index 1. If the user configures either of these, the default transform will not be used.

When neither of these optional phase1 proposals is set, the default transform will be used in the phase 1 proposal. This default transform has the following settings:

Encryption Algorithm: 3DES CBC
Hash Algorithm: SHA1
Diffie-Hellman Group: 2
SA Lifetime seconds: 86400 seconds (=1 day)
SA Lifetime kBytes: 0x10000 kByte (=64 MByte)

The default authentication method is **preshared key** unless a local **certificate** has been specified in the CHIP.INI file.

If this API is used to set both of the phase 1 transforms available for user control, then the authentication method specified in these two transforms must be set the same. Otherwise unpredictable operation may result.

## Beck C-Library Usage

**IPsec_IKE_Phase1_Set()**

## Related Topics

**IPSEC_IKE_PHASE1_DELETE**
**IKE_PH1_AGGRESSIVE** definition in CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xCF09: IPSEC_IKE_PHASE1_DELETE

Remove one of the two optional user phase 1 IKE transforms.

## Parameters

*AX*

0xCF09 (= IPSEC_IKE_PHASE1_DELETE)

*CX*

Index 0 or 1 selects which of two phase 1 IKE transforms is to be deleted.

## Return Value

DX = 0, AX = **error code**.

## Comments

The specified phase 1 transform is removed. If both user settable phase 1 transforms are not present, then the **default** transform will be used.

## Beck C-Library Usage

**IPsec_IKE_Phase1_Delete()**

## Related Topics

**IPSEC_IKE_PHASE1_SET**
**IKE_PH1_AGGRESSIVE** definition in CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAC service 0xCF0A: IPSEC_RESTORE_POLICY

Restore policy and/or preshared keys from file.

## Parameters

*AX*

0xCF0A (= IPSEC_RESTORE_POLICY)

*ES:BX*

Pointer to path of **binary** file containing the IP security policies and any desired preshared keys.

## Return Value

DX = 0, AX = 0 on success

On failure AX indicates error:

-3 : Policy file exceeded 64 Kbyte size limit
-4 : Work buffer memory allocation failed
-5 : Invalid policy file contents
202 : Policy file open failed

Between 1000 and 1999 :

Policy add operation failed. Subtract 1000 from this error code and look **here**for resulting error code description.

Between 2000 and 2999:

Preshared key add operation failed. Subtract 2000 from this error code and look **here**for resulting error code description.

## Comments

This API allows IP security policy and/or preshared keys to be set based on the contents of a binary policy file in same manner as done with the **POLICY_FILE** CHIP.INI file entry.

## Beck C-Library Usage

**IPsec_Restore_Policy()**

## Related Topics

**IPSEC_ADD_POLICY**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

End of document

## SSL_HandshakeComplete

Check handshake status.

```
int SSL_HandshakeComplete(int sd,
                          unsigned int timeout);
```

## Parameters

*sd*

Socket descriptor

*timeout*

Block period in milliseconds:
0xFFFF for block forever
0 for no blocking (polling)
1 through 65534 milliseconds maximum wait.

## Return Value

status:
-3: Handshake failure
-2: Invalid socket descriptor
-1: Not a TCP type socket
0: Either not an SSL type socket or handshake in progress
1: SSL handshake completed

## Comments

Use of this function is optional. It allows the user to determine whether or not the SSL handshake process has completed and that the data socket is now ready for data transmission and reception of user data.

This API can be used in either polling mode (`timeout` = 0) or blocking mode. Blocking mode will return as soon as the status indication return value is non-zero, or the specified time-out period has elapsed. The `timeout` parameter may be set to 0xFFFF to block for an indefinite amount of time, in which case the return value is guaranteed to be non-zero on return.

Alternatively, the user can simply make repeated calls to the **send()** and/or **recv()** functions until this handshake is completed. The SSL handshake is performed in the user's thread, so one of these three functions must be called in order to complete the SSL connection.

## See Also

SSL_SessionNew  send()  recv()

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.10 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.07 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

# @CHIP-RTOS - Flash write/erase cycles
# Important information about IPC@CHIP® drives and file system
# (Limited Number of Flash erase cycles per sector)

---

The internal Flash sectors of the IPC@CHIP® (used for drive A: and memory for @CHIP-RTOS code) have limited erase cycles. So you should avoid cyclic write access like writing into a log file every hour.

We guarantee 10.000 erase cycles per sector on SC1x and 100.000 erase cycles per sector on SC1x3. Note that one write access in your program (fwrite or write) can cause more than one write/erase access on the @CHIP's flash disk.

**Optimizations:**

To decrease the flash write accesses in your program, you should follow the hints below:

1. Use fopen, fwrite, ...
2. Use setvbuf with a size of 1024 bytes.

**Description:**

fopen internally works with a 512 byte stream buffer. The file system sectors have a size of 512 bytes. Flash physical sector size on SC1x can be 256 or 1024 bytes, depending on the flash memory used. Setting the stream buffered size to 1024 helps to avoid additional flash writes.
Flash physical sector size on SC1x3 is 64 kbytes. The internal Flash Translation Layer (FTL) on SC1x3 uses wear leveling to makes sure that data writes are evenly distributed over the whole storage media.

Note:

> The fwrite function in Borland C 5.02 has an unexpected behaviour when the size of data exceeds the buffered size (512 bytes default or size set with setvbuf). In this case the data blocks exceeding the buffered size are always written in an unbuffered manner instead of splitting them into smaller packets.

**How a flash defect can be detected:**

The return code tells you if a write access was not successful. If the failure was due to a flash disk defect there will be an error message

> "Flash error at sector: xx"

in the console text at the next system boot. Also the execution of the **AUTOEXEC.BAT** will be bypassed in this case.

**Workaround:**

If you need to generate a periodic log file, you should use the **RAM** disk to store the file. To save the file when the power goes down, you could use the **Powerfail Interrupt (PFI)** feature.

Another option is to use an IDE Drive (Hard disk, Compactflash, ..) to store the log file. How to connect an IDE Drive to the IPC@CHIP® is explained in the API **documentation**.

---

End of document

---

# @CHIP-RTOS - External Disk Interface

## External Disk Drive

Here is the interface definition for an external disk B: and D: drive. This interface allows you to add an external B: or D: drive. This drive must be block (sector) oriented. Each sector should be 512 bytes long. The application must provide a software interrupt 0xB1 function for drive B: or 0xB2 for drive D: to read and write these sectors on this drive.

On SC1x systems, the maximum disk size is about 2 Gigabytes. (This limitation is due to the FAT16 disk format used on SC1x systems.) SC1x systems support only drive B:

The SC1x3/SC23 systems support FAT32 format. The maximum disk size on SC1x3/SC23 systems is about 8 Terabytes. The maximum file size is 2**32 bytes minus one cluster (=32 kbyte), which is about 4 Gigabyte. SC1x3/SC23 systems support drive B: and drive D:.

### Interrupt 0xB0 Services

Service is selected by index in AH register.


- **0x01: Install External Disk B:**
- **0x02: Deinstall External Disk B:**
- **0x03: Install a notification callback function**
- **0x04: Uninstall the notification callback function**
- **0x05: Install External Disk D:**
- **0x06: Deinstall External Disk D:**


---

## Interrupt 0xB0 service 0x01: Install External Disk B:

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    Sector number (BX is MSH of unsigned long)
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.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xB0 service 0x02: Deinstall External Disk B:

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?)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xB0 service 0x03: Install a notification callback function

The external drives B: and D: may be hot-plugable. This means that the drive medium might be removed or inserted while your application is running. In some cases it may be important for you to be notified if a medium is removed or inserted, for example if one of your tasks has set its working directory to the external drive. By default @CHIP-RTOS will reset the working directory of all tasks that previously had their working directory on a drive that has been removed to A:\. However this might not be appropriate. So a reaction on the removal of a hot-pluggable medium might be to set a new valid working directory. To be notified of insertion and removal events, install an EXTDISK callback function.

A maximum of 12 EXTDISK callback functions can be installed.

The following events may be passed along with the callback function's `event` parameter:

EXTDISK_EVENT_INSTALL_B = 0x01
Drive B: has been installed

EXTDISK_EVENT_REMOVE_B = 0x02
Drive B: has been uninstalled

EXTDISK_EVENT_INSTALL_D = 0x03
Drive D: has been installed

EXTDISK_EVENT_REMOVE_D = 0x04
Drive D: has been uninstalled

## Parameters

*AH*
0x03

*ES:DI*
Pointer to the callback function of the type EXTDISK_CALLBACK

EXTDISK_CALLBACK is defined as follows:

```
typedef void (far * EXTDISK_CALLBACK)(unsigned int event);
```

## Return Value

AX =0 success
AX !=0 failed

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xB0 service 0x04: Uninstall the notification callback function

Uninstall the notification callback function

## Parameters

*AH*
0x04

*ES:DI*
Pointer to the callback function of the type EXTDISK_CALLBACK (see Install a notification callback function for details)

## Return Value

AX =0 success
AX !=0 failed

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

Top of list
Index page

---

## Interrupt 0xB0 service 0x05: Install External Disk D:

This function will logically install a D: drive.
Prior to this call the application must provide a disk read/write function **installed** at interrupt 0xB2. This installed handler function should expect:

```
AX       1 for write, 0 for read.
BX,DX    Sector number (BX is MSH of unsigned long)
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*
0x05

*BX,DX*
An unsigned long with the total number of sectors. BX holds high word.

### Return Value

AX is 0 if OK.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xB0 service 0x06: Deinstall External Disk D:

Used to deinstall drive D: which was installed before with function 0x05 (above).

### Parameters

*AH*
0x06

### Return Value

AX =0 success

AX !=0 failed (may be drive was not installed?)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

End of document

---




# @CHIP-RTOS C Library - RTOS API

Header File: `#include <RTOS.H>`
which includes `<RTXAPI.H>`

Task Control

Semaphores

Timing

Events

Messages

## Task Control Functions

| | |
|---|---|
| RTX_Create_Task | Create and start a task |
| RTX_Param_Task | Create and start a task with parameters |
| RTX_Create_Task_Without_Run | Create a task |
| RTX_Param_Task_Without_Run | Create a task with parameters |
| RTX_Delete_Task | Delete a task |
| RTX_Kill_Task | Stop and kill a task |
| RTX_Restart_Task | Start execution of task |
| RTX_End_Execution | End execution of current task |
| RTX_Sleep_Time | Sleep for a defined time |
| RTX_Sleep_Fine | Sleep for a number of RTI |
| RTX_Sleep_Long | Sleep for a long number of RTI |
| RTX_Sleep_Request | Sleep until wakeup request |
| RTX_Wakeup | Wakeup a sleeping task |
| RTX_Suspend_Task | Suspend execution of a task |
| RTX_Resume_Task | Resume execution of a suspended task |
| RTX_Get_TaskID | Get the ID of the current task |
| RTX_Change_TaskPrio | **Deprecated!** Change specified task's priority |
| RTX_Task_Priority | Read/write specified task's priority |

| | |
|---|---|
| RTX_Disable_Task_Scheduling | Disable task switching |
| RTX_Enable_Task_Scheduling | Re-enable task switching |
| RTX_Access_Filesystem | Reserve file system access by task |
| -- Task Monitoring -- | |
| RTX_Get_Task_List | Get list of current tasks |
| RTX_Get_Task_State | Get state of a task |
| RTX_Get_Task_State_Ext | Get state of a task |
| RTX_Start_Task_Monitor | Install task monitor. |
| RTX_Stop_Task_Monitor | Disable task monitoring |
| RTX_Count_Resources | Count available/used resources |
| RTX_Wait_Queue | Diagnostic routine |

## Semaphore Functions

| | |
|---|---|
| RTX_Create_Sem | Create a semaphore |
| RTX_Delete_Sem | Delete a semaphore |
| RTX_Find_Sem | Find semaphore by name |
| -- For either type Semaphore -- | |
| RTX_Wait_Sem_Priority | Wait with specified priority |
| -- For Counting Semaphores -- | |
| RTX_Get_Sem | Get a counting semaphore, no wait |
| RTX_Wait_Sem | Wait on a counting semaphore |
| RTX_Signal_Sem | Signal a counting semaphore |
| -- For Resource Semaphores -- | |
| RTX_Reserve_Sem | Reserve a resource semaphore |
| RTX_Free_Sem | Set use count to zero |
| RTX_Release_Sem | Decrement use count by 1 |

## Timing Functions

| | |
|---|---|
| -- Time/Date -- | |
| RTX_Get_TimeDate | Get system time and date |
| RTX_Set_TimeDate | Set system time and date |
| -- System Time -- | |
| RTX_Get_System_Ticks | Get tick count of system clock |
| RTX_GetTickCount | Return tick count of system clock |
| RTX_GetTick_us | Return system clock microseconds |
| RTX_GetFineTick | Read system clock with high resolution |
| RTX_FineTickInit | Initialize for RTX_GetTickCount() use |
| RTX_Get_RTI_Ticks | Get Real-Time Interrupt count |
| RTX_Get_RTI_Rate | Get Real-Time Interrupt rate |

-- RTX Timer Callbacks --

| | |
|---|---|
| RTX_Install_Timer | Install timer procedure |
| RTX_Install_TimerP | Install timer procedure (alternate method) |
| RTX_Fine_Timer | Install high resolution timer |
| RTX_Fine_TimerP | Install high resolution timer (alternate method) |
| RTX_Remove_Timer | Remove timer procedure |
| RTX_Start_Timer | Start periodic execution of timer procedure |
| RTX_Stop_Timer | Stop periodic execution of timer procedure |
| RTX_Timer_Delay | Set timer delay milliseconds |
| RTX_Timer_Delay_RTI | Set timer delay RTI ticks |
| RTX_Install_RTI_Hook | Install callback to execute in real-time interrupt |

## Event Functions

| | |
|---|---|
| RTX_Create_EventGroup | Create event group |
| RTX_Delete_EventGroup | Delete an event group |
| RTX_Signal_Events | Signal events group |
| RTX_Wait_For_Event | Wait for events |
| RTX_Wait_For_EventP | Wait for events (alternate method) |
| RTX_Get_EventGroup_State | Read current events on a group |
| RTX_Get_Saved_Events | Get the saved event flags/bits |
| RTX_Find_EventGroup | Find an event group |

## Message Functions

| | |
|---|---|
| RTX_Create_Msg | Create message exchange |
| RTX_Delete_Msg | Delete a message exchange |
| RTX_Send_Msg | Send a message to a message exchange |
| RTX_Get_Msg | Get a message from a message exchange, no wait |
| RTX_Wait_For_Msg | Wait for a message, optional timeout |
| RTX_Wait_For_MsgP | Wait for a message, optional timeout (alternate method) |
| RTX_Find_Msg | Find a message exchange by name |

List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS - RTOS Overview

IPC@CHIP® Documentation **Index**

## RTOS Overview

RTOS API The RTOS **API** services split into the following groups:

- **Task_Control Services**
- **Semaphore_Services**
- **Time_/ Date Services**
- **Timer_Procedures**
- **Event_Manager**
- **Message_Exchange Manager**

## Task Control Services

**RTX_Create_Task()** Create and start a task
**RTX_Param_Task()** Create and start a task with parameters
**RTX_Create_Task_Without_Run()** Create a task
**RTX_Param_Task_Without_Run()** Create a task with parameters
**RTX_End_Execution()** End execution of a task
**RTX_Kill_Task()** Kill a task
**RTX_Restart_Task()** Trigger start of task execution
**RTX_Delete_Task()** Remove a task from the system
**RTX_Get_TaskID()** Get ID of the current task
**RTX_Change_TaskPrio()** Change the priority of a task
**RTX_Sleep_Time()** Sleep for a specified time
**RTX_Sleep_Long()** Sleep for a long time
**RTX_Sleep_Fine()** Sleep for a specified RTI count
**RTX_Sleep_Request()** Sleep until a wake request
**RTX_Wakeup()** Wake a task which is sleeping
**RTX_Suspend_Task()** Suspend a task
**RTX_Resume_Task()** Resume a task
**RTX_Access_Filesystem()** Enable file access for the calling task
**RTX_Get_Task_State()** Get state of a task (task monitoring)
**RTX_Get_Task_State_Ext()** Get state of a task
**RTX_Get_Task_List()** Get list of tasks in system
**RTX_Start_Task_Monitor()** Start task monitoring
**RTX_Stop_Task_Monitor()** Stop task monitoring

# Semaphore Services

Semaphores are used to guarantee a task mutually exclusive access to a critical resource. 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 @CHIP-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 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 the **RTX_Create_Sem()**API. The @CHIP-RTOS needs a unique four byte semaphore name and on success returns a new semaphore ID (or handle) to the caller. This handle is needed for the 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 to **RTX_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 **RTX_Wait_Sem()**or **RTX_Get_Sem()**. If the resource is available the @CHIP-RTOS gives it to the task immediately. When the task is finished using the resource, it signals its release by calling **RTX_Signal_Sem()**.

**Using a resource semaphore:**

A resource semaphore is created by specifying an initial count of -1 in the call of **RTX_Create_Sem()**. The @CHIP-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 **RTX_Reserve_Sem()**with the semaphore ID returned by **RTX_Create_Sem()**. The resource is released with a call to **RTX_Release_Sem()**.

**Semaphore Services:**

**RTX_Create_Sem()** Create a semaphore
**RTX_Delete_Sem()** Delete a semaphore
--- Resource Semaphore API ----
**RTX_Free_Sem()** Free a resource semaphore
**RTX_Release_Sem()** Release a resource semaphore
**RTX_Reserve_Sem()** Get a resource semaphore
--- Counting Semaphore API ----
**RTX_Get_Sem()** Get use of a counting semaphore (no wait)
**RTX_Signal_Sem()** Signal a counting semaphore

**RTX_Wait_Sem()** Wait on a counting semaphore (optional timeout)

**Top of list**
**Index page**

---

## Time / Date Services

The following time/date and clock services are available.

**RTX_Get_TimeDate()** Get system time and date
**RTX_Set_TimeDate()** Set system time and date
**RTX_Get_System_Ticks()** Get system clock ms (output param)
**RTX_GetTickCount()** Return system clock ms
**RTX_Get_RTI_Ticks()** Get RTI tick count
**RTX_GetTick_us()** Get system time in us
**RTX_GetFineTick()** Get high resolution system time

### Related Topics

TimeDate_Structure **type** definition

**Top of list**
**Index page**

---

## Timer Procedures

The @CHIP-RTOS provides the following API for the usage of timer callback procedures:

**RTX_Install_Timer()** Install a timer procedure
**RTX_Fine_Timer()** Install a high resolution timer procedure
**RTX_Remove_Timer()** Remove a timer procedure
**RTX_Start_Timer()** Start periodic execution of callback
**RTX_Stop_Timer()** Stop periodic execution of callback
**RTX_Timer_Delay()** Set delay time for next callback

The @CHIP-RTOS implements timer procedures using kernel objects, of which there are a limited number. The sum of semaphores plus event groups plus timer procedures is limited to this maximum **count**since all three of these objects require use of a kernel object.

The kernel executes periodic user timer procedures at a specified time interval. Timer callbacks execute in the system's top priority AMXK kernel task. Consequently, your timer procedure must be as short as practical without any waiting or endless loops in order to not interfere with other system activities. Avoid the usage of large C-library functions such as `printf()`. However, any of the following kernel services are reasonable to call from within a timer procedure:

- **RTX_Remove_Timer()**
- **RTX_Start_Timer()**
- **RTX_Stop_Timer()**
- **RTX_Timer_Delay()**
- **RTX_GetTickCount()**
- **RTX_GetTick_us()**

- RTX_GetFineTick()
- RTX_Wakeup()
- RTX_Signal_Sem()
- RTX_Signal_Events()
- RTX_Send_Msg()
- RTX_Suspend_Task()
- RTX_Resume_Task()

This list is not exhaustive.

Top of list
Index page

---

## Event Manager

The internal @CHIP-RTOS Event Manager provides a convenient mechanism for coordinating tasks waiting for events. The events can be signalled by other tasksor timerprocedures. The 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.

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 number of event groups which can be created is limited by the total numberof kernel objects available. One kernel object is expended to create a semaphore, event group or timer procedure.

The Event Manager is useful 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 Manger then 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:**

RTX_Create_EventGroup() Create an event group
RTX_Delete_EventGroup() Delete an event group
RTX_Signal_Events() Signal one or more events in a group
RTX_Wait_For_Event() Wait for all/any of a set of events in a group
RTX_Get_EventGroup_State() Read current state of events in a group
RTX_Get_Saved_Events() Get saved event flags
RTX_Find_EventGroup() Find Event Group by name

Top of list
Index page

---

## Message Exchange Manager

The @CHIP-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 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. The maximum depth of a mailbox is specified when the Message Exchange is created, which indicates the maximum number of messages which can be stored in that particular mailbox.

Messages are delivered to the mailboxes of the Message Exchange in message envelopes. The system's maximum number of available messages envelopes is 64 (or 128 for SC1x3/SC23). The maximum number of message exchanges is 10 (or 32 for SC1x3/SC23). The maximum message length is 12 bytes. (Note: Larger messages can be implemented with a pointer and a length parameter in the message.) 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 the message 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 maximum time that it is willing to wait for a message. When more than one message is available at the exchange, a task requesting a message will be given the message at the head of the higest priority non-empty mailbox FIFO (of which there are four).

The task also specifies a priority at which it should receive the next message, allowing either for that task to cut in line ahead of other tasks that were waiting at the same message exchange or to yield to other tasks that later come looking for a message with a higher wait priority (lower numeric value). So conceptually, there are 2**16 FIFO priority queues at each Message Exchange where tasks wait for a message. When a message becomes available, this message will be handed to the first task waiting in the highest priority non-empty task wait queue.

What can be confusing here is that there are two concepts of priority involved here with Message Exchanges (neither of which is related to **task priority**).

1. Message Priority - Ranges 0 through 3 (identifies which mailbox)
2. Wait Priority - Ranges 0 through 0xFFFF (identifies priority queue)

**Message Exchange Services:**

**RTX_Create_Msg()** Create a Message Exchange
**RTX_Delete_Msg()** Delete a Message Exchange
**RTX_Send_Msg()** Send a message to a Message Exchange
**RTX_Get_Msg()** Get a message, if any (no wait)
**RTX_Wait_For_Msg()** Wait for message to arrive (optional timeout)
**RTX_Find_Msg()** Find a Message Exchange, specified by name

**Top of list**
**Index page**

---

End of document

---




# @CHIP-RTOS - RTOS Error Codes



## @CHIP-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

---




# @CHIP-RTOS - RTOS Application Developers Note



## @CHIP-RTOS Application Notes

RTOS

RTOS **API** Listing

### Comments

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!!

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 may block other important tasks of our system. e.g. the serial and Ethernet receiver tasks.

End of document




# @CHIP-RTOS - RTOS Examples Available



## RTOS Examples

RTOS

Available RTOS **API** examples:

1. `taskexp1.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.

The C example programs use the **Beck C-Library** and so should you if you are programming in C or C++.

End of document

# @CHIP-RTOS - Data Structures used in RTOS API

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** :


- **RTX_COUNTS**
- **RTX_DETAILS**
- **RTX_DETAIL_EVENT**
- **RTX_DETAIL_MSGXCHG**
- **RTX_DETAIL_SEM**
- **RTX_DETAIL_TASK**
- **RTX_DETAIL_TMR**
- **RTX_Msg**
- **RTX_Wait_Event**
- **RTX_Wait_Msg**
- **TaskDefBlock**
- **TASK_PARAM_DEF**
- **TaskList**
- **Task_StateData**
- **TimeDate_Structure**
- **TimerProc_Structure**


## RTX_COUNTS

```
typedef struct tag_rtx_counts
{
    unsigned int    nTask_Avail ;       // Tasks control blocks available.
    unsigned int    nTask_In_Use ;      // Tasks control blocks in use.

    unsigned int    nObj_Avail ;        // RTX objects available.
    unsigned int    nSem_In_Use ;       // Number of semaphores existing.
    unsigned int    nTimer_In_Use ;     // Number of timers existing.
    unsigned int    nEventGrp_In_Use ;  // Number of Event Groups existing.

    unsigned int    nMsg_Xchg_Avail ;   // Number of Msg Exchanges available.
    unsigned int    nMsg_Xchg_In_Use ;  // Number of Msg Exchanges in use.

    unsigned int    nEnvelop_Avail ;    // Number of Msg Envelopes available.
    unsigned int    nEnvelop_In_Use ;   // Number of Msg Envelopes in use.

    unsigned int    nReport_Cnt ;       // Number of objects report if all
                                        // unfiltered objects were reported.
    unsigned int    nReport_Bytes ;     // Size in bytes report would be if all
                                        // unfiltered objects were reported.
    unsigned int    Internal_Err ;      // Should be zero, else RTX unhealthy.
    unsigned int    TCB_Q_Offset ;      // Wait Q offset in Task Control Block
    unsigned int    RTX_Data_Segment ;  // Segment where TCB reside

} RTX_COUNTS ;
```

## Comments

This structure is defined in the header file rtxapi.h. A data structure of this form provided by the user is filled with data by the **RTX_Count_Resources()** API.

**nTask_Avail**

Number of task control blocks which are available to the task creation API, **RTX_Param_Task()**.

**nTask_In_Use**

The current number of tasks present in system.

**nObj_Avail**

Number of RTX objects free for use by: timer creation API such as **RTX_Install_Timer()**, semaphore creation API **RTX_Create_Sem()**, or event group creation API **RTX_Create_EventGroup()**. A generic RTX object supports these three system object types.

**nSem_In_Use**

The current number of semaphores present in the system.

**nTimer_In_Use**

The current number of RTX timers present in the system.

**nEventGrp_In_Use**

The current number of Event Groups present in the system.

**nMsg_Xchg_Avail**

Number of **Message Exchanges**which can yet be created with the **RTX_Create_Msg()**API.

**nMsg_Xchg_In_Use**

The current number of Message Exchanges present in system.

**nEnvelop_Avail**

Number of message envelopes available for the **RTX_Send_Msg()**API. These containers are required for storing the message at the exchange in the case that no task is waiting for the message.

**nEnvelop_In_Use**

The current number of message envelopes awaiting pick up at the Message Exchanges.

**nReport_Cnt**

Total number of system objects which would be reported in the `nReport_Bytes`report when sufficient buffer space is provided.

**nReport_Bytes**

Total number of bytes required to report all the **RTX_DETAILS**objects that the **RTX_Count_Resources()**API would report, given sufficient space in the `details`buffer. If this API is called with `list_size`greater or equal to this count, then all requested data has been reported.

**Internal_Err**

This value should always remain zero. A non-zero value indicates that an internal inconsistency was found inside the RTX private data.

**TCB_Q_Offset**

This constant is the offset into the Task Control Blocks (TCB) at which the wait queues reside. These circularly linked list are used to queue tasks awaiting a Message Exchange, Semaphore or Event Group. This constant is reported here since it is subject to change in future @Chip-RTOS releases.

**RTX_Data_Segment**

This is the data segment at which the RTX has its private data. Pointers to Task Control Blocks (TCB) can be formed as:

```
tcb_ptr = MK_FP(RTX_Data_Segment, task_id) ;
```

## Related Topics

API function **RTX_Count_Resources()** - Count RTX resources

**Top of list**
**Index page**

---

# RTX_DETAILS

```
typedef union tag_rtx_detail
{
    void                *pBuf ;
    RTX_DETAIL_SEM      *pSem ;          // Semaphore details
    RTX_DETAIL_TMR      *pTimer ;        // Timer details
    RTX_DETAIL_EVENT    *pEvent ;        // Event details
    RTX_DETAIL_TASK     *pTask ;         // Task details
    RTX_DETAIL_MSGXCHG  *pMsgXchg ;      // Message Exchange details

} RTX_DETAILS ;

    ////////////////////////////////////////////////////
    // Example usage of RTX_DETAILS union
    ////////////////////////////////////////////////////
void Analyze_RTX_Usage (unsigned int list_size, unsigned int filter)
{
    void *buffer = malloc(list_size) ;
    if (buffer)
    {
        int entries = RTX_Count_Resources(&Counts,
                                          buffer,
                                          list_size,
                                          filter) ;
        RTX_DETAILS details.pBuf = buffer ;     // For scanning
        for (i = 0 ; i < entries ; i++)
        {
            switch (details.pSem->Type)
            {
            case KRTX_SEM_TYPE:
                Process_Sem(details.pSem) ;
                details.pSem++ ;
                break ;
            case KRTX_TIMER_TYPE:
                Process_Timer(details.pTimer) ;
                details.pTimer++ ;
                break ;
            case KRTX_EVENT_GROUP_TYPE:
                Process_Event_Group(details.pEvent) ;
                details.pEvent++ ;
                break ;
            case KRTX_MSG_XCHG_TYPE:
                Process_Msg_Xchg(details.pMsgXchg) ;
                details.pMsgXchg++ ;
                break ;
            case KRTX_TASK_TYPE:
                Process_Task(details.pTask) ;
                details.pTask++ ;
                break ;
            default:
                i = entries ;
                printf("ERROR: Invalid Type - 0x%02X\r\n",
                        details.pSem->Type) ;
                break ;
            }   // End switch
        }       // End FOR loop

        free (buffer) ;
    }
}
```

## Comments

This union is defined in the header file rtxapi.h. It can be used to parse the buffer output by the **RTX_Count_Resources()** API.

In the example code shown above, the RTX_DETAILS union of pointers is used to scan the sequence of data structures of varying size output by the **RTX_Count_Resources()** API. The number of bytes that the buffer pointer is advanced on each pass through the while loop depends on the data structure type recognized.

Each of the non-void data structures referenced by the pointers in this union have a `Type` member located at a common offset within the structures. This member is an enumerator which has one of the following values to indicate the type data structure present.

0 KRTX_SEM_TYPE for **semaphore**details
1 KRTX_TIMER_TYPE for RTX **timer**details
2 KRTX_EVENT_GROUP_TYPE for **Event Group**details
3 KRTX_MSG_XCHG_TYPE for **Message Exchange**details
4 KRTX_TASK_TYPE for **task**details

## Related Topics

API function **RTX_Count_Resources()** - Count RTX resources

**Top of list**
**Index page**

---

# RTX_DETAIL_EVENT

```
typedef struct tag_rtx_detail_event
{
    unsigned int  Handle ;          // Event Group "ID"
    char          szName[5] ;       // Nil terminated 4 char name
    char          Type ;            // Enumerator = KRTX_EVENT_GROUP_TYPE
    unsigned int  TaskQHead ;       // Task ID of task awaiting msg.
    unsigned int  Current_Events ;  // Event Group bits state

} RTX_DETAIL_EVENT ;
```

## Comments

This data structure is defined in the header file rtxapi.h. The **RTX_Count_Resources()** API uses this structure type to report details about an Event Group that is in use within the system.

**`Handle`**
Event Group ID as output by **RTX_Create_EventGroup()**API.

**`szName`**
Zero terminated 4 character name. This may not necessarily be ASCII. Event Groups are not required to have a name.

**`Type`**
Set to KRTX_EVENT_GROUP_TYPEto distinguish data structure types within the **RTX_DETAILS**union.

**`TaskQHead`**
If no tasks are waiting on this Event Group, then this value matches `Handle`(an empty circularly linked list). Otherwise this references a linked list node within the Task Control Block of the first task waiting on this Event Group. Subtracting the `TCB_Q_Offset`value reported in the **RTX_COUNTS**data structure yields the task ID of the waiting task.

**`Current_Events`**
The current state of the Event Group.

## Related Topics

API function **RTX_Count_Resources()** - Count RTX resources
RTOS **Event** Manager

**Top of list**
**Index page**

---

# RTX_DETAIL_MSGXCHG

```
typedef struct tag_rtx_detail_msgxchg
{
    unsigned int  Handle ;      // Message Exchange "ID"
    char          szName[5] ;   // Nil terminated 4 char name
    char          Type ;        // Enumerator = KRTX_MSG_XCHG_TYPE
    unsigned int  TaskQHead ;   // Task ID of task awaiting msg.
    unsigned char Busy ;        // xchg busy flag
    unsigned char AvailFlags ;  // Message available flags
    // Following arrays have an entry for each message priority
    unsigned int  Slots[4];     // Msg slots available
    unsigned int  MaxMsg[4] ;   // Maximum number of messages
} RTX_DETAIL_MSGXCHG ;
```

## Comments

This data structure is defined in the header file rtxapi.h. The **RTX_Count_Resources()** API uses this structure type to report details about a Message Exchange that is in use within the system.

**`Handle`**
Message Exchange ID as output by **RTX_Create_Msg()**API.

**`szName`**
Zero terminated 4 character name. This may not necessarily be ASCII. Event Groups are not required to have a name.

**`Type`**
Set to KRTX_MSG_XCHG_TYPEto distinguish data structure types within the **RTX_DETAILS**union.

**`TaskQHead`**
If no tasks are waiting on this Message Exchange, then this value matches `Handle`(an empty circularly linked list). Otherwise this references a linked list node within the Task Control Block of the first task waiting on this Message Exchange. Subtracting the `TCB_Q_Offset`value reported in the **RTX_COUNTS**data structure yields the task ID of the waiting task.

**`Busy`**
Counter non-zero when kernel is currently manipulating the Message Exchange object.

**`AvailFlags`**
This is a bit field indicating a message is available at each of the four priorities. BN is set if there is a message available at priority N, where N is in range 0..3.

**`Slots`**
This array has an entry for each message priority in the range 0..3. This count indicates the number of remaining message slots at the respective priority. When this count reaches zero, the **RTX_Send_Msg()**API will fail due to "mailbox full" condition.

**MaxMsg**

This array has an entry for each message priority in the range 0..3. This count indicates the maximum number of message slots available at the respective message priority. Note that the number of messages currently posted on the exchange at priority 'i' can be computed as:

```
unsigned int msg_cnt = MaxMsg[i] - Slots[i] ;
```

## Related Topics

API function **RTX_Count_Resources()** - Count RTX resources
RTOS **Message** Exchange Manager

**Top of list**
**Index page**

---

# RTX_DETAIL_SEM

```
typedef struct tag_rtx_detail_sem
{
    unsigned int  Handle ;      // Semaphore "ID"
    char          szName[5] ;   // Nil terminated 4 char name
    char          Type ;        // Enumerator = KRTX_SEM_TYPE
    unsigned int  TaskQHead ;   // Task ID of task awaiting msg.
    unsigned char Use_Count ;   // Semaphore use counter
    unsigned char Attribute ;   // B0=1 if resource type semaphore
    int           Value ;       // Semaphore up/down counter
    unsigned int  Owner ;       // ID of task owning this semaphore

} RTX_DETAIL_SEM ;
```

## Comments

This data structure is defined in the header file rtxapi.h. The **RTX_Count_Resources()** API uses this structure type to report details about a semaphore that is in use within the system.

**Handle**

Semaphore ID as output by **RTX_Create_Sem()**API.

**szName**

Zero terminated 4 character name. This may not necessarily be ASCII. Semaphores are not required to have a name.

**Type**

Set to KRTX_SEM_TYPEto distinguish data structure types within the **RTX_DETAILS**union.

**TaskQHead**

If no tasks are waiting on this semaphore, then this value matches `Handle`(an empty circularly linked list). Otherwise this references a linked list node within the Task Control Block of the first task waiting on this semaphore. Subtracting the `TCB_Q_Offset`value reported in the **RTX_COUNTS**data structure yields the task ID of the waiting task.

**Use_Count**

Resource semaphore's use up/down counter. For resource semaphores this count is set to 1 when

a task first acquires the semaphore and is incremented on each additional **reservation**made by that task. This count is positive when the semaphore is in use. For resource type semaphores, this count is set to 1 when the semaphore is in use.

**Attribute**
This is a bit field with two bits defined:

0x1 - KRTX_RESOURCE_SEMA resource type semaphore
0x2 - KRTX_SEM_BUSYSemaphore in use by system

**Value**
This is an up down counter used to represent the semaphore's state. For counting semaphores this count is positive when the semaphore has been **signaled**and is available. The count decrements with each **RTX_Wait_Sem()**or **RTX_Get_Sem()**call. N tasks are waiting on this counting semaphore when the value is negative N.

For resource type semaphores this value is 1 when semaphore is available and zero when it is in use.

**Owner**
For resource type semaphores this is the task ID of the RTX task which currently owns the semaphore, if non-zero.

## Related Topics

API function **RTX_Count_Resources()** - Count RTX resources
RTOS **Semaphore** Services

**Top of list**
**Index page**

---

# RTX_DETAIL_TASK

```
typedef struct tag_rtx_detail_task
{
    unsigned int  Handle ;      // Task "ID"
    char          szName[5] ;   // Nil terminated 4 char name
    char          Type ;        // Enumerator = KRTX_TASK_TYPE
    unsigned int  WaitQ ;       // Node in list of tasks awaiting an object.
    unsigned int  Status ;      // Task status
    unsigned int  WaitParam ;   // Semaphore handle, if awaiting semaphore.
    unsigned int  WaitParam2 ;  // Msg Xchg handle, if awaiting message.

} RTX_DETAIL_TASK ;
```

## Comments

This data structure is defined in the header file rtxapi.h. The **RTX_Count_Resources()** API uses this structure type to report details about a task within the system.

**Handle**
Task ID as output by **RTX_TASK_CREATE**API.

**szName**

Zero terminated 4 character name. This may not necessarily be ASCII. Tasks are not required to have a name, although it is the convention.

**WaitQ**
If this task is currently waiting on an RTX object, then this vector references the next task in the queue. This linked list forms a circular queue which eventually leads back to the object on which the task is waiting (semaphore, Event Group or Message Exchange). When this value refers to a task in the queue, then the `TCB_Q_Offset`value reported in the **RTX_COUNTS**data structure must be subtracted from this value to obtain that task's ID. No offset is subtracted out if the next object in list is the semaphore, Event Group or Message Exchange.

**Status**
This is the task state bit field, same as reported for the **RTX_Get_Task_State()**API. Refer to the **taskState**specification for detailed definition.

**WaitParam**
This RTX internal data can mean different things at different times. When the task is parked at a semaphore, this will be the handle to that semaphore.

**WaitParam2**
This RTX internal data can mean different things at different times. When the task is parked at a Message Exchange, this will be the handle to that Message Exchange.

## Related Topics

API function **RTX_Count_Resources()** - Count RTX resources

**Top of list**
**Index page**

---

# RTX_DETAIL_TMR

```
typedef struct tag_rtx_detail_tmr
{
    unsigned int  Handle ;      // Timer "ID"
    // Note: @Chip-RTOS timers have no 4 letter name.
 RTX_TIMER_CALLBACK Callback ;  // Vector to timer's callback.
    char          fStopped ;    // Timer stopped
    char          Type ;        // Enumerator = KRTX_TIMER_TYPE
    unsigned long Period ;      // [ms]
    unsigned long Param ;       // Timer procedures parameter

} RTX_DETAIL_TMR ;
```

## Comments

This data structure is defined in the header file rtxapi.h. The **RTX_Count_Resources()** API uses this structure type to report details about a RTX timer that is in use within the system.

**Handle**
Timer ID as output by **RTX_Install_Timer()**API.

**Callback**
Far vector to timer's user callback function.

**fStopped**
Boolean, zero if timer is currently running. Set to 1 if timer is stopped.

**Type**
Set to KRTX_TIMER_TYPEto distinguish data structure types within the **RTX_DETAILS**union.

**Period**
Timers interval in milliseconds.

**Param**
The optional long parameter passed to the callback procedure.

## Related Topics

API function **RTX_Count_Resources()** - Count RTX resources

**Top of list**
**Index page**

---

# RTX_Msg

```
typedef struct tag_rtx_msg{
  unsigned 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;
  int  mb3;

} RTX_Msg;
```

## Comments

This structure is defined in the header file rtxapi.h.

Prior to making the **RTX_Create_Msg()** call, the caller must set the following members of the `RTX_Msg` data structure.

**name**
Here you can give the Message Exchange a unique four character name. This field is optional. If you plan to use the **RTX_Find_Msg()**API then you should provide a unique name here for all Message Exchanges.

**mb0** through **mb3**
State here the maximum number of message envelopes to be queued at each of this Message Exchange's four priority mailboxes. The system's total number of available messages envelopes is 64 (since @CHIP-RTOS version 1.02B, else 32). For SC1x3/SC23 the total message envelopes is extended to 128.

The `msgID` member is an output parameter from the **RTX_Create_Msg()** API call.

## Related Topics

RTOS **Message** Exchange Manager

# 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.

Prior to making the **RTX_Wait_For_Event()** call, the caller must set the members of the `RTX_Wait_Event` data structure.

**`mask`**
Event bits set to '1' in this mask indicate the events in `value`which will be matched. All other event bits in the group are ignored.

**`value`**
The desired event state for the event bits set to '1' in `mask`are stated here. User may wait on an event bit transition to either '1' or '0', as specified here.

**`match`**
This is a Boolean value which indicates whether you require all indicated events to match (AND, `match`!= 0), or whether it is sufficient that only a single event of those specified match (OR, `match`=0).

**`timeout`**
Maximum time to wait for events, specified in milliseconds. Zero here indicates wait forever. Negative values indicate to return immediately with a *timed out*indication if the events are not ready (event polling).

## Related Topics

API function **RTX_Wait_For_Event()** - Wait for events in a group
RTOS **Event** Manager



# RTX_Wait_Msg

```
typedef struct tag_rtx_wait_msg{
  unsigned int msgID; // ID of the Message Exchange
  unsigned int prio;  // Priority for wait [0 .. 0xFFFF], 0 = highest
  void far *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 calling the **RTX_Wait_For_Msg()** ALI, 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_For_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. (Note that this priority has no connection to either **task**priority or Message send **mailbox**priority.)

**`msg`**
Put here a far pointer to a 12 byte buffer allocated in your application program memory. The Message Exchange Manager will copy the message to this buffer if a message is available. The format of the 12 byte object referenced by `msg`is user defined.

**`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. Negative values are not permitted.

## Related Topics

API function **RTX_Wait_For_Msg()** - Wait for a message from a Message Exchange
RTOS **Message** Exchange Manager

**Top of list**
**Index page**

---

# TaskDefBlock

```
typedef struct tag_taskdefblock
{
        void     (far *proc)();    // Task entry vector (far)
        char           name[4];    // Task name, 4 characters not null terminated
   unsigned int far *stackptr;     // Task stack pointer (far)
        unsigned int   stacksize;    // size of stack (bytes)
        unsigned short attrib;       // task attributes, TA_SAVE_STACK bit
        short int      priority;     // task priority,  range: 2..127 inclusive
        unsigned short time_slice;   // 0: none, !=0: number of milliseconds before task
                                     // is forced to relinquish processor
        short       mailboxlevel1;   // First parameter passed to task
        short       mailboxlevel2;   // Second parameter passed to task
        short       mailboxlevel3;   // not used
        short       mailboxlevel4;   // not used

} TaskDefBlock;
```

## Comments

This data structure is deprecated. Please use the **TASK_PARAM_DEF** data structure instead.

This structure definition remains in the C-library header file rtxapi.h for compatibility with older code. If you do not need your task to accept any input parameters then you are still free to use this compatible data structure with the **RTX_Create_Task()** API-call, however at a slight additional cost due to the larger data structure.

The new and old task definition data structures are identical up to the `mailboxlevel1` field, which was previously unused. Refer to the **TASK_PARAM_DEF** documentation for descriptions of the member fields.

If you use the new TASK_PARAM_DEF structure with the old RTX_Create_Task() API-call, you have to use a cast to suppress warnings that the compiler will generate due to the diffrent type.

```
TASK_PARAM_DEF taskParameters;
.
.
.
RTX_Create_Task( &taskId, (TaskDefBlock *)&taskParameters );
```

## Related Topics

API function **RTX_Create_Task()** - Create and start a task
**TASK_PARAM_DEF** - Alternate data structure

**Top of list**
**Index page**

---

# TASK_PARAM_DEF

```
typedef void huge (far *PARAM_TASK)(unsigned short Param1, unsigned short Param2) ;

typedef struct tag_TASK_PARAM_DEF
{
        PARAM_TASK    proc ;            // task procedure pointer
        char    name[4];                // name 4 characters not null terminated
        unsigned int far * stackptr;    // task stack pointer (Top of stack)
        unsigned int stacksize;         // size of task stack (bytes)
        unsigned short int attrib;      // task attributes, TA_SAVE_STACK bit
        short int priority;             // task priority,  range: 2 <= priority <= 127
        unsigned short int time_slice;  // 0: none, !=0: number of milliseconds before task
                                        //  force to relinquish processor
        unsigned short Param1 ;         // FP_OFF portion if a far pointer is used
        unsigned short Param2 ;         // FP_SEG portion if a far pointer is used

} TASK_PARAM_DEF ;
```

## Comments

This structure is defined in the header file rtxapi.h.

Prior to making the **RTX_Param_Task()** API call, the caller must set each of the `TASK_PARAM_DEF` structure members.

**`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(...)`
Microsoft C: `void far _saveregs _loadds taskfunc(...)`

These declarations assure that the data segment register (DS) is loaded on entry into your task. Where the three dots are shown in these prototypes, you must replace these with your desired parameters to be passed to your task by the system. You are free to choose void parameters. Here are a few possible parameter passing options you could use:

```
taskfuncA(void)
taskfuncB(unsigned int param1, unsigned int param2)
taskfuncC(void far *this_object)
taskfuncD(struct MY_STRUC_TYPE far *this_object)
taskfuncE(struct MY_STRUC_TYPE near *this_object)
taskfuncF(unsigned long combined_param)
```

All above examples except for `taskfuncB()` would need to be cast to `PARAM_TASK` type as they are assigned to the `proc` member. The parameters cannot exceed 32 bits in total, which is the amount of data that the system pushes onto the stack for task input parameters. See `Param1` and `Param2` **members**.

**`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 you task's stack space (highest address). Your tasks 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 pushing data onto 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 then 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.

**`attrib`**

This is a bit field with currently one option attribute defined. The amount of user task stack space used by the system prior to calling the task procedure can be reduced by 20 bytes by setting the B0 bit to 1 (see TA_SAVE_STACKdefine). This setting is recommended here unless your task requires backwards compatibility to older @Chip-RTOS versions (not very likely) with respect to this user stack preparation done by the system prior to calling your task entry procedure.

On entry into the task from the system, the stack situation will be as follows:

Stack Fence WORD 2 = 0x5555 Highest address
Stack Fence WORD 1 = 0x5555
Task name characters 3 & 4
Task name characters 1 & 2
Pad bytes (20 bytes if TA_SAVE_STACKbit == 0, else 0 bytes)
Param2 16 bit WORD
Param1 16 bit WORD
FP_SEG(return address)
FP_OFF(return address) <-- SS:SP pointing here

**`priority`**

Application program tasks can range in priority from 2 to 127 (inclusive), where 2 is higher priority. Generally, task priorities between 20 and 30 are recommended. This recommendation is based on the priority assignments of the built-in **system**tasks. Too high a priority for an application task may block important system tasks: e.g. the Ethernet receiver task. However, in some cases higher priority application tasks are appropriate design, but in these cases you must keep the execution dwell of these high priority task very short. User DOS applications are started at priority 25.

**Note:**

Internally all tasks have a unique priority. When a task is **created**or its priority is **changed**, that task is given a lower internal task priority than any other task in the system with the same user task priority.

**`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. Time slicing is disabled on all DOS application main tasks.

**SC1x3/SC23 Note:**

If the **RTI**rate has been **configured**for rates higher than 1000 Hz, than this time-slice count will be translated to RTI ticks from milliseconds within the @Chip-RTOS task creation API. Any time-slice values in milliseconds exceeding 65535 real-time interrupts will be limited to 65535 RTI ticks.

**`Param1`**

This will be the left most parameter declared in your task procedure if two 16 bit values are used as input parameters. This would be the FP_OFF() portion of a far pointer input parameter or least

significant half of a long 32 bit input parameter.

**Param2**
This will be the right most parameter declared in your task procedure if two 16 bit values are used as input parameters. This would be the FP_SEG() portion of a far pointer input parameter or most significant half of a long 32 bit input parameter.

### Related Topics

API function **RTX_Param_Task()** - Create and start a task
**TaskDefBlock** - Legacy data structure

**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
    unsigned int taskCount;   // Duty cycle indicator
    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. Task **Monitoring** mode must be enabled to obtain all the data listed here.

**`taskID`**

Task handle used for the RTX API functions.

**`taskPrio`**

Current task **priority**.

**`taskState`**

This bit field is zero for active tasks. The reasons for an inactive task waiting are coded as follows:

B0: Timer wait (used in combination with other bits: B2, B3, B4, B7)
B1: Trigger wait (i.e. idle, see note)
B2: Semaphore wait
B3: Event Group wait
B4: Message Exchange wait
B5: -- not implemented --
B6: Suspended (waiting for resume)
B7: Asleep waiting for wakeup
B8 - B15 internal use only

Notes:

- A task is in "trigger wait" state prior to starting (e.g. **RTX_Create_Task_Without_Run()**) or after termination. A **RTX_Restart_Task()** call exits this state.
- Where as B1, B2, B3, B4 and B7 are mutually exclusive wait conditions, B6 "**Suspended**" condition can be added (OR'ed in) to these other wait conditions.

**`taskCount`**

This value, ranging from 0 to 10000, provides a rough indication of the amount load placed on the system by this task. At each execution of the @CHIP-RTOS real-time interrupt handler (1000 Hz **rate**by default), this count is incremented if this particular task is either currently executing or if the CPU is idle and this task was the most recent task to be executed. The running count is recorded at 10 second intervals (assuming 1000 Hz operation), so a task which is active 100% of the time would score a `taskCount`of 10000.

**`stackused`**

This value is the percentage of the task's stack space used. The system presets stack space to all zeroes prior to starting tasks. The deepest (lowest address) non-zero value on the stack indicates the "high water mark". This value is only meaningful for tasks created within application programs. DOS programs normally switch stacks on entry, so this the stack space usage measurement is defeated.

**`stacksize`**

This is the total number of bytes in the task's stack space. This value is only meaningful for tasks created within application programs, due to that DOS programs normally switch stacks on entry.

## 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
{
   unsigned int  far *timerID;  // pointer to storage the unique timerID
   void (far *proc)();          // pointer to the procedure to be executed
   void  far *dummyptr;         // Optional parameter to timer procedure
   char  name[4];               // -- Field not used --
   long  interval;              // timer execution interval in milliseconds
                                //  (or RTI ticks if RTX_TIMER_FINE API is used)
} 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 set as specified here:

**`timerID`**
Put here a far pointer to a 16 bit location in your program's memory space. The **RTX_Install_Timer()**API 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. Have also a look to **some important notes**concerning Timer Procedures.

**`dummyptr`**

This four byte value is pushed onto the stack by the system as an input parameter to your timer procedure. You are free to ignore this parameter as suggested by the above `MyTimerProc(void)`declarations. Alternatively you could also use, for example, either of the following timer procedure declarations (Borland):

```
void huge MyTimerProc(unsigned int timer_id,
                      unsigned long lParam) ;
```

or

```
void huge MyTimerProc(unsigned int timer_id, void far *ptr) ;
```

The `timer_id`parameter will be the same as output to `timerID`location in this data structure. In either case, the second parameter is a copy of the four byte `dummyptr`.

**`name`**

This provisional field is not used by the system.

**`interval`**

Specify here the interval at which you want your timer procedure to be called. For the **RTX_Install_Timer()**API this interval is specified in milliseconds. For the SC1x3/SC23 **RTX_Fine_Timer()**API this count is specified in Real-Time Interrupt (RTI) ticks, which differs from milliseconds when the **RTI**rate is **configured**for frequencies above 1000 Hz.

## Related Topics

API function **RTX_Install_Timer()** - Install a timer procedure
API function **RTX_Fine_Timer()** - Install a high resolution timer procedure
RTOS **Timer** Procedures

Top of list
Index page

End of document

## IPC@CHIP® System Tasks

RTOS API

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
NMI      priority= 1     Non-Maskable Interrupt task
TCPT     priority= 4     TCP/IP timer task
ETH0     priority= 5     Ethernet receiver task
PPPS     priority= 6     PPP server (PPPSERVER ENABLE)
PPPC     priority= 6     PPP client (PPPCLIENT ENABLE)
CFGS     priority= 7     UDP config server
TELN     priority= 11    Telnet server (TELNET ENABLE)
MTSK     priority= 12    Console task (command shell)
WEBS     priority= 41    Web server (WEB ENABLE)
FTPS     priority= 41    FTP server (FTP ENABLE)
SSLW     priority= 41    Secure Socket Layer Web server (SC1x3/SC23 only)
(SSL ENABLE)
```

## Related Topics

API function **RTX_Param_Task()** - Create and start a task

## RTX_Sleep_Time

Go to sleep for a specified number of milliseconds.

```
int RTX_Sleep_Time ( unsigned int time_ms );
```

## Parameters

*time_ms*

Sleep time in milliseconds. A zero delay time is translated to 1 millisecond within this API.

## Return Value

0 on success else **error** code.

## Comments

The **RTX_Wakeup** function can wake up a sleeping task before its sleep timer has expired.

If the task has an outstanding wake request pending when it calls here, the task will continue execution immediately without waiting.

For sleep times over 65 seconds, see **RTX_Sleep_Long** API.

## See Also

**RTX_Sleep_Request** **RTX_Wakeup** **RTX_Sleep_Fine**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Create_Task

Create and start a task

```
int RTX_Create_Task ( int far *taskID,
                      const TaskDefBlock far *taskdefblock );
```

## Parameters

*taskID*

Output Parameter: Pointer to 16 bit storage for the taskID, allocated by the caller

*taskdefblock*

Input Parameter: Pointer to a **TaskDefBlock** type data structure prepared by caller.

## Return Value

0, task is running, output location `taskID` contains the 16 bit taskID
non-zero --> **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_Create_Task` ). This function differs with **RTX_Create_Task_Without_Run** in that the new task is released to run within this function call.

You may want to use the **helper_alloc_rtos_mem** API to allocate the new task's stack space. You can then optionally use the **helper_assign_mem** API to reassign this stack space memory allocation to the new task. This step would be taken after this `RTX_Create_Task` API has provided you the new task ID. Assigning the stack memory allocation to the task has the advantage that this stack memory will then be released automatically by the system when this new task is later deleted from the system.

**Caution:** For small or medium memory model, the task's stack allocation will need to be made from the program data segment in order that the model's DS == SS flat segment requirement be met. This means that the RTOS memory allocation cited here will not be a good idea for these memory models. Although small or medium memory models provide more efficient programs than large memory model, the large memory model is highly recommended for all users interested in getting their program up and running without a lot of adventure.

## See Also

RTX_Create_Task_Without_Run RTX_Delete_Task helper_alloc_rtos_mem helper_assign_mem

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

IPC@CHIP® System **Tasks**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Param_Task

Create and start a task which is passed parameters by system.

```
int RTX_Param_Task ( int *taskID,
                     const TASK_PARAM_DEF *taskdefblock );
```

## Parameters

*taskID*

Output Parameter: Pointer to 16 bit storage for the taskID, allocated by the caller

*taskdefblock*

Input Parameter: Pointer to a **TASK_PARAM_DEF** type data structure prepared by caller.

## Return Value

0, task is running, output location `taskID` contains the 16 bit taskID
non-zero --> **error** code.

## Comments

The caller must fill in portions of the **TASK_PARAM_DEF** 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_Param_Task` ).

This function differs with **RTX_Param_Task_Without_Run** in that the new task is released to run within this function call.

This API is identical to the **RTX_Create_Task**, with one difference. The task definition data structure used here allows a 32 bit parameter to specified which the system passes to the task on the stack. Any of the following would be valid prototypes for your task:

```
void huge My_TaskA(unsigned short Param1,
                   unsigned short Param2) ;
void huge My_TaskB(unsigned short Param1) ;
void huge My_TaskC(void far *this_object) ;
void huge My_TaskD(void near *this_object) ;
```

Each of these tasks, except for `My_TaskA` , would need to be cast to type `PARAM_TASK` when assigned to the `proc` member of the **TASK_PARAM_DEF** structure. For My_TaskC case, the following settings for `Param1` and `Param2` **members** in the `TASK_PARAM_DEF` structure would be required so that the far

pointer `this_object` parameter would point to `My_Object` :

```
Task_Def.Param1 = FP_OFF(&My_Object) ;
Task_Def.Param2 = FP_SEG(&My_Object) ;
```

For My_TaskD case, the pointer to the near `My_Object` can be set using only the first 16 bit parameter as follows:

```
Task_Def.Param1 = (unsigned short)((void near *)&My_Object) ;
```

In the case where your task only uses a single 16 bit parameter, the system has pushed the second parameter onto the stack as well but it is of no consequence to your task.

## See Also

**RTX_Param_Task_Without_Run** **RTX_Create_Task** **RTX_Delete_Task**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

IPC@CHIP® System **Tasks**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Create_Task_Without_Run

Create a task which is passed parameters by system, but do not start the task yet.

```
int RTX_Create_Task_Without_Run ( int far *taskID,
                          const TaskDefBlock far *taskdefblock );
```

## Parameters

*taskID*

Output Parameter: Pointer to 16 bit storage for the taskID, allocated by the caller

*taskdefblock*

Input Parameter: Pointer to a **TaskDefBlock** type data structure prepared by caller.

## Return Value

0, task is created, output location `taskID` contains the 16 bit taskID
non-zero --> **error** code.

## Comments

The caller must **fill** in portions of the **TaskDefBlock** structure prior to making this call.

The **RTX_Restart_Task** function must be called to start this new task.

## See Also

**RTX_Create_Task** **RTX_Delete_Task** **RTX_Restart_Task**
**RTX_Param_Task_Without_Run**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

IPC@CHIP® System **Tasks**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Param_Task_Without_Run

Create a task which will be passed a parameter, but do not start it yet.

```
int RTX_Param_Task_Without_Run ( int *taskID,
                          const TASK_PARAM_DEF *taskdefblock );
```

## Parameters

*taskID*

Output Parameter: Pointer to 16 bit storage for the taskID, allocated by the caller

*taskdefblock*

Input Parameter: Pointer to a **TASK_PARAM_DEF** type data structure prepared by caller.

## Return Value

0, task is created, output location `taskID` contains the 16 bit taskID
non-zero --> **error** code.

## Comments

The caller must fill in portions of the **TASK_PARAM_DEF** structure prior to making this call.

The **RTX_Restart_Task** function must be called to start this new task.

This API is identical to the **RTX_Create_Task_Without_Run**, with one difference. The task definition data structure used here allows a 32 bit parameter to specified which the system passes to the task on the stack.

## See Also

**RTX_Param_Task** **RTX_Delete_Task** **RTX_Restart_Task**
**RTX_Create_Task_Without_Run**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

IPC@CHIP® System **Tasks**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Kill_Task

Kill the specified task. The killed task remains in the system and can be restarted.

```
int RTX_Kill_Task ( int taskID );
```

## Parameters

*taskID*

Task handle returned from **RTX_Create_Task** or **RTX_Create_Task_Without_Run**.

## Return Value

0, success
non-zero --> **error** code.

## Comments

This function will force a ready, executing or suspended task to end. All requests for task execution pending at the time of the kill request will be erased.

A task can kill itself.

You should not delete or kill a task, which is waiting, or is about to wait for a resource or counting semaphore.

## See Also

**RTX_End_Execution** **RTX_Delete_Task** **RTX_Restart_Task**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Delete_Task

Remove specified task from system.

```
int RTX_Delete_Task ( int taskID );
```

## Parameters

*taskID*

Task handle returned from **RTX_Create_Task** or **RTX_Create_Task_Without_Run**.

## Return Value

0, task is removed.
non-zero --> **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.
CAUTION:

Any memory allocated from the system by the task which is being deleted here will be released.

## See Also

**RTX_Create_Task** **RTX_Create_Task_Without_Run** **RTX_Kill_Task**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Get_TaskID

Get ID of the currently executing task.

```
int RTX_Get_TaskID ( void );
```

## Parameters

-- none --

## Return Value

task ID.

## See Also

RTX_Create_Task

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## RTX_Sleep_Request

The calling task will sleep until some other task makes a **RTX_Wakeup** call.

```
int RTX_Sleep_Request ( void );
```

## Parameters

-- none --

## Return Value

0: Always success.

## Comments

If the task has an outstanding wake request pending when it calls here, the task will continue execution immediately without waiting.

## See Also

**RTX_Wakeup** **RTX_Sleep_Time**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Wakeup

Wake up a task expected to be waiting within a RTX_Sleep_Request or RTX_Sleep_Time call.

```
int RTX_Wakeup ( int taskID );
```

## Parameters

*taskID*

Task handle returned from RTX_Create_Task or RTX_Create_Task_Without_Run.

## Return Value

0, success
non-zero --> error code.

## Comments

If the task is not currently sleeping, then the RTOS sets a "wakeup pending" flag for the respective task such that on the next call the task makes to one of the sleep functions it will return ("wake up") immediately.

An immediate task switch will occur if the task being wakened is of higher priority than the current task.

## See Also

RTX_Sleep_Request  RTX_Sleep_Time

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_End_Execution

End execution of current task. This call can only be used to terminate the task which is making the call.

```
int RTX_End_Execution ( void );
```

## Parameters

-- none --

## Return Value

There is no return from this function.

## Comments

A call to this function is equivalent to a return from the task's main procedure.

The task can later be restarted with the **RTX_Restart_Task()** API.

## See Also

**RTX_Delete_Task** **RTX_Restart_Task** **RTX_Kill_Task**
**RTX_Suspend_Task**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## RTX_Change_TaskPrio

Change specified task's priority.

```
int RTX_Change_TaskPrio ( int taskID, int priority,
                          int *error );
```

This function is **deprecated**! Use RTX_Task_Priority instead.

## Parameters

*taskID*

Task handle returned from RTX_Create_Task or RTX_Create_Task_Without_Run.
Set to zero for current task.

*priority*

Task priority in range 2...127 where priority 2 is high and 127 is low.

*error*

Output Parameter: Pointer to error variable

## Return Value

0, success.
non-zero --> Variable error contains error code.

## Comments

The calling tasks' priority can be adjusted by setting the `taskID` parameter to zero. The AMXK kernel task's priority cannot be changed.

An out of range `priority` value will be limited to range 2..127 inside this function. (An exception is the value 0x5ABC for which no operation will be performed.)

## See Also

RTX_Get_TaskID RTX_Disable_Task_Scheduling RTX_Task_Priority

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS C Library - RTOS API

## RTX_Task_Priority

Read out existing and/or set a task's priority.

```
int RTX_Task_Priority ( int taskID, int priority);
```

## Parameters

*taskID*

Task handle returned from **RTX_Create_Task** or **RTX_Create_Task_Without_Run**, or set to zero to indicate the current task.

*priority*

Task priority in range 2...127 where priority 2 is high and 127 is low, or set to `PRIORITY_READ_ONLY` (=0x5ABC) to make no change to the task priority and only read out the existing task priority.

## Return Value

non-negative, success: Existing task priority (prior to any change)
negative, failure: **error** code, such as -1 for "Invalid task ID".

## Comments

Set the `priority` parameter to `PRIORITY_READ_ONLY` if you only want to read out the task's priority without making any change.

The calling tasks' priority can be read out or adjusted by setting the `taskID` parameter to zero. The **AMXK** kernel task's priority cannot be changed.

An out of range `priority` value will be limited to range 2..127 inside this function.

## See Also

**RTX_Get_TaskID** **RTX_Disable_Task_Scheduling** **RTX_Change_TaskPrio**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Access_Filesystem

Reserve file access context space for the calling task.

```
int RTX_Access_Filesystem ( void );
```

## Parameters

-- none --

## Return Value

0, success
-1 --> Failure (Too many tasks with file access)

## Comments

This API is deprecated. (However, it may still have some diagnostic value to determine why file accesses are failing or could be used to acquire the file access context space ahead of time to assure the task's future file accesses will succeed.)

It is no longer necessary that a task use this API to gain access to the file system. A task is automatically granted file system access when the file system API is used, provided that the system has not ran out of file access context space.

The file system supports registration of up to 15 tasks at a time. Each DOS program's main task is registered with the file system regardless of whether or not that program uses the file system. A task's reservation on the file system access context space remains in effect until that task is deleted from the system. The DOS program main tasks are deleted when the program terminates (or **terminates and stays resident**).

Once a successful call to this API has been made, any further calls perform no operation (at most one context slot is assigned per task).

If this API returns a failure indication, then the number of tasks registered with the file system has reached the limit and this calling task is consequently not being granted access to the file system.

## See Also

**RTX_Create_Task**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Get_Task_State

Get state of a task specified by name using the task monitoring mode.

```
int RTX_Get_Task_State ( const char far *taskname,
                         Task_StateData far *taskdata,
                         int *error );
```

## Parameters

*taskname*

Input parameter, pointer to 4 character unique name tag of the task whose state information is desired.

*taskdata*

Output parameter, pointer to a `Task_StateData` **type** structure, allocated by the user which will receive data.

*error*

Output parameter, pointer to 16 bit word to receive error code indicator: 0 on success, else **error** code.

## Return Value

-1 ==> Failure, error code was output to `error` In this case error code 0 means that **Task Monitoring** is not enabled.
Other ==> Task ID

## Comments

The task monitoring mode must be enabled in order to use this function.

## See Also

**RTX_Get_Task_State_Ext** **RTX_Start_Task_Monitor**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Get_Task_List

Get list of current tasks in system.

```
int RTX_Get_Task_List ( TaskList far *tasklist, int length );
```

## Parameters

*tasklist*

Output parameter: Pointer to an array of `TaskList` **type** structures allocated by user to be filled by this function

*length*

Length of the list, number of data structures in array.

## Return Value

Number of tasks output to `tasklist` array

## Comments

The caller must allocate sufficient buffer space at `tasklist` to allow all tasks to be reported, including those **created** by the system in order to receive a complete report.

## See Also

**RTX_Get_Task_State** **RTX_Count_Resources**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Start_Task_Monitor

Install task monitor.

```
void RTX_Start_Task_Monitor ( void );
```

## Parameters

-- none --

## Return Value

-- none --

## Comments

The 1000 Hz system clock interrupt is intercepted to perform task monitoring.

## See Also

RTX_Stop_Task_Monitor

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## RTX_Stop_Task_Monitor

Disable task monitoring.

```
void RTX_Stop_Task_Monitor ( void );
```

## Parameters

-- none --

## Return Value

-- none --

## Comments

This function restores the system's original 1000 Hz interrupt handler, removing the task timing function installed by **RTX_Start_Task_Monitor**.

## See Also

**RTX_Start_Task_Monitor**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document




# @CHIP-RTOS C Library - RTOS API

---

## RTX_Suspend_Task

Suspend the execution of a task until **RTX_Resume_Task** is called to resume the task.

```
int RTX_Suspend_Task ( int taskID );
```

## Parameters

*taskID*

Task ID of task to be suspended.

## Return Value

0 on success else **error** code, invalid task ID.

## See Also

**RTX_Resume_Task** **RTX_End_Execution**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Resume_Task

Enable the execution of a suspended task

```
int RTX_Resume_Task ( int taskID );
```

## Parameters

*taskID*

Task ID of task to be resumed.

## Return Value

0 on success else **error** code, invalid task ID.

## See Also

**RTX_Suspend_Task**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Restart_Task

Start a newly created task or restart a killed task.

```
int RTX_Restart_Task ( int taskID );
```

## Parameters

*taskID*

Task handle returned from **RTX_Create_Task** or **RTX_Create_Task_Without_Run**.

## Return Value

0 on success else **error** code, invalid task ID.

## See Also

**RTX_Create_Task_Without_Run** **RTX_Kill_Task**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Get_Task_State_Ext

Get state of a task specified by name without the task monitoring mode.

```
int RTX_Get_Task_State_Ext ( const char far *taskname,
                             unsigned int far *taskstate,
                             int *error );
```

## Parameters

*taskname*

Input parameter: Pointer to 4 character unique name tag of the task whose state information is desired.

*taskstate*

Output parameter: Pointer to 16 bit word which will receive task state bit field defined below.

*error*

Output parameter: Pointer to 16 bit word to receive error code indicator: 0 on success, else **error** code.

## Return Value

-1 ==> Failure, error code was output to `error`
Other ==> Task ID

## 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

## See Also

RTX_Get_Task_State RTX_Count_Resources

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Count_Resources

This diagnostic routine reads out the number of RTX resources available and optionally obtains detailed data about the resources that are in use.

```
int RTX_Count_Resources (RTX_COUNTS far *counts,
                         void       far *details,
                         unsigned int     list_size,
                         unsigned int    filter );
```

## Parameters

*counts*

Output parameter: Pointer to an **RTX_COUNTS** data structure that will be filled by this API.

*details*

Optional Output parameter: Pointer to a buffer which will receive objects which can be referenced with the **RTX_DETAILS** union of structure pointers. This API will report various details for each RTX object to this buffer. This parameter can be set to NULL if no detailed RTX object data is desired.

*list_size*

Number of bytes in buffer at `details` . If either this value is zero or `details` is NULL, then no `RTX_DETAILS` reporting is done.

*filter*

Optional output listing filter. This bit field selects which type RTX objects are listed in detailed report at `details` . Set this to zero to report all object types.

## Return Value

-1: Pointer `counts` was not valid.
>= 0: Number of **RTX_DETAILS** structures output to array at `details` . If this count is lower than the `nReport_Cnt` member of the **RTX_COUNTS** output to `counts` then the report was not complete due to insufficient buffer space (`list_size` less than `nReport_Bytes` member of `RTX_COUNTS` ).

## Comments

The primary purpose for this API is to allow the user to determine the number of RTX resources that are available. For this limited purpose, only the **RTX_COUNTS** data structure is required at the location referenced by the `counts` parameter. The `details` parameter can be set to NULL or `list_size`

parameter set to zero to indicate that no **RTX_DETAILS** reporting is desired.

When more detailed information about the various RTX objects in use is desired, an array of **RTX_DETAILS** unions can be provided by the caller at location specified by the `details` pointer. The number of bytes space available in this buffer is specified in the `list_size` argument. This API will then report details for as many of the selected RTX objects as will fit into this buffer space.

The report `filter` parameter allows you to focus the detailed reporting on particular types of RTX objects. This can simplify your memory management problem by reducing the size of the `RTX_DETAILS` array at `details` required to report all the objects of the type you are interested in. Bits set to one in this bit field inhibit the reporting of the RTX object type associated with that bit. The following filter masks are defined in RTXAPI.H header file for this purpose:

RTX_FILT_SEM - Enables semaphore detail reporting
RTX_FILT_TIMER - Enables timer detail reporting
RTX_FILT_EVENT_GROUP - Enables Event Group detail reporting
RTX_DETAIL_MSGXCHG - Enables Message Exchange detail reporting
RTX_DETAIL_TASK - Enables task detail reporting

Each of these mask has all bits set except for one. These masks can be combined with the bit wise AND operation to report more than one object type. For example:

```
RTX_FILT_SEM & RTX_FILT_EVENT_GROUP
```

for the `filter` argument would report both semaphore and Event Group objects.

One possible strategy for allocating the `details` buffer memory would be to first call this API with `list_size` set to zero only to discover how large a buffer will be required to collect the desired detailed information. The number of buffer bytes required is reported by the API in the `nReport_Bytes` member of the **RTX_COUNTS** data structure. This reported count is influenced by what ever reporting `filter` you had specified.

Inside this API, the RTX data is collected with the **task switching** locked. Interrupts are enabled after inhibiting task switching. The interrupt mask is restored to the state on entry prior to re-enabling the task switching on return. Nevertheless, a task switch can occur at the moment task switching is re-enabled here, thereby defeating any interrupt mask set by the caller.

## See Also

**RTX_Wait_Queue** **RTX_Get_Task_State**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Wait_Queue

This diagnostic routine reads out the list of tasks waiting at a specified RTX semaphore, Event Group or Message Exchange.

```
int RTX_Wait_Queue ( unsigned int handle,
                     void       far *details,
                     unsigned int     list_size,
                     unsigned int    filter );
```

## Parameters

*handle*

Input parameter: RTX object identifier for a semaphore, Event Group, or Message Exchange whose wait list is desired.

*details*

Optional Output parameter: Pointer to a buffer which will receive objects which can be referenced with the **RTX_DETAILS** union of structure pointers. The first object in this list will apply to the RTX object specified by `handle` . All subsequent reported objects will be **RTX_DETAIL_TASK** type, one for each task waiting in the queue listed from first to last. This parameter can be set to NULL in order to receive a return value indicating how large the buffer should be for a subsequent call.

*list_size*

Number of bytes in buffer at `details` . If either this value is zero or `details` is NULL, then no `RTX_DETAILS` reporting is done and the return value indicates with a negative value how large a buffer will be required for a subsequent call.

*filter*

Must be one of the following masks to indicate to what type of RTX resource the `handle` applies:

RTX_FILT_SEM - For semaphore
RTX_FILT_EVENT_GROUP - For an Event Group
RTX_FILT_MSGXCHG - For a Message Exchange

## Return Value

Less than zero: Either `details` was NULL or more than `list_size` will be required for reporting the requested data. In this case, the negative of this return value is the minimum size buffer required for the report.
Equal zero: Error, most likely the `handle` did not apply to an RTX resource of the type indicated by `filter` .

Greater than zero: Number of **RTX_DETAILS** structures output to array at `details`. The first structure applies to the RTX object referenced by `handle`. The subsequent entries are all **RTX_DETAIL_TASK** type. The number of task waiting at the queue is equal to this return value minus one.

## Comments

This diagnostic API is used to determine which tasks are waiting in a queue at an RTX semaphore, Event Group or Message Exchange. The `filter` argument must indicate to which type RTX resource the `handle` argument applies.

Inside this API, the RTX data is collected with the **task switching** locked. Interrupts are enabled after inhibiting task switching. The interrupt mask is restored to the state on entry prior to re-enabling the task switching on return. Nevertheless, a task switch can occur at the moment task switching is re-enabled here, thereby defeating any interrupt mask set by the caller.

## See Also

**RTX_Count_Resources** **RTX_Get_Task_State**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Create_Sem

Create either a counting or resource **semaphore**.

```
int RTX_Create_Sem ( int far *semID, const char far *name,
                     int initvalue );
```

## Parameters

*semID*

Output Parameter: pointer to where semaphore ID will be written.

*name*

Input Parameter: pointer to a four character name for the new semaphore. This string need not be zero terminated, but must contain four bytes.

*initvalue*

New semaphore's initial value. Set this to -1 to create a resource semaphore. For counting semaphore's set this to some non-negative number representing the number of resources initially available on this new semaphore.

## Return Value

0 on success else **error** code.

## Comments

The semaphore name is only relevant if the **RTX_Find_Sem** service will be used. Otherwise, setting this field to all zeros can avoid namespace conflicts with other semaphores which do use the name lookup service.

## See Also

**RTX_Delete_Sem** **RTX_Find_Sem**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Delete_Sem

Delete a semaphore, removing it from system.

```
int RTX_Delete_Sem ( int semID );
```

## Parameters

*semID*

The semaphore ID of a resource or counting semaphore acquired by a call to **RTX_Create_Sem**.

## Return Value

0 on success else **error** code.

## Comments

You must be absolutely certain that no other task, ISP or Timer Procedure is in any way using or about to use the semaphore. Failure to observe this restriction may lead to unexpected and unpredictable faults.

## See Also

**RTX_Create_Sem**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## RTX_Free_Sem

Unconditionally free a resource semaphore. The resource's usage count is set to zero and the resource is freed.

```
int RTX_Free_Sem ( int semID );
```

## Parameters

*semID*

The semaphore ID of a resource semaphore acquired by a call to **RTX_Create_Sem**.

## Return Value

0 on success else **error** code.

## Comments

The resource will immediately be given to the task (if any) which is waiting at the head of this semaphore's wait queue. Task rescheduling occurs immediately if necessary.

This function is similar to **RTX_Release_Sem**. The RTX_Release_Sem function unwinds the usage count from nested calls to **RTX_Reserve_Sem** one count per call. This RTX_Free_Sem function clears the count, immediately erasing the affect of several nested calls to RTX_Reserve_Sem.

## See Also

**RTX_Release_Sem** **RTX_Reserve_Sem**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Get_Sem

Attempt to get use of a counting semaphore, without waiting.

```
int RTX_Get_Sem ( int semID );
```

## Parameters

*semID*

The semaphore ID of a counting semaphore acquired by a call to **RTX_Create_Sem**.

## Return Value

0 on success (semaphore granted) else **error** code.

## Comments

This call returns with an error if the semaphore is in use (represented as semaphore count zero). Otherwise the semaphore count is decremented by one and the caller is granted the semaphore.

## See Also

**RTX_Signal_Sem** **RTX_Wait_Sem**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## RTX_Release_Sem

Release a resource semaphore.

```
int RTX_Release_Sem ( int semID );
```

## Parameters

*semID*

The semaphore ID of a resource semaphore acquired by a call to **RTX_Create_Sem**.

## Return Value

0 on success else **error** code.

## Comments

The resource's usage count is decremented by one for each call to this function. The resource is not freed until the usage count becomes zero.

Once freed, the resource will immediately be given to the task (if any) which is waiting at the head of this semaphore's wait queue. Task rescheduling occurs immediately if necessary.

## See Also

**RTX_Free_Sem** **RTX_Reserve_Sem**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

End of document

## RTX_Reserve_Sem

Reserve a resource semaphore. If current task is already the owner of this semaphore then the semaphore's usage count is incremented by one and the function returns immediately with success.

```
int RTX_Reserve_Sem ( int semID, long far *time_ms );
```

## Parameters

*semID*

The semaphore ID of a resource semaphore acquired by a call to **RTX_Create_Sem**.

*time_ms*

Pointer for ms to wait: Zero -> Wait forever for semaphore
Positive value -> Maximum number of milliseconds to wait for resource
Negative value -> Polling mode, return immediately. If semaphore was not available the return value will be the -27 time-out indication.

## Return Value

0 on success else **error** code.

## Comments

The calling task will be suspended waiting for the resource if the resource is owned by some other task.

When the resource is finally available for use by the caller, the procedure will return with 0.

Upon first acquiring ownership of the resource, the resouce's usage count is set to one.

It is permissible for a task to call `RTX_Reserve_Sem` even if the calling task already owns the resource. The resource usage count is incremented by one for each subsequent call by the resource owner.

The wait priority used here is 20. (See the **RTX_Wait_Sem_Priority** alternative API for explanation of wait priority.)

## See Also

**RTX_Release_Sem** **RTX_Free_Sem** **RTX_Wait_Sem_Priority**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Signal_Sem

Signal a counting semaphore.

```
int RTX_Signal_Sem ( int semID );
```

## Parameters

*semID*

The semaphore ID of a counting semaphore acquired by a call to **RTX_Create_Sem**.

## Return Value

0 on success else **error** code.

## Comments

The resource will immediately be given to the task (if any) which is waiting at the head of this semaphore's wait queue. Task rescheduling occurs immediately if necessary.

This method is used to release access to a counting semaphore after a task has acquired the semaphore through a **RTX_Get_Sem** or **RTX_Wait_Sem** call.

## See Also

**RTX_Get_Sem** **RTX_Wait_Sem**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

End of document

## RTX_Wait_Sem

Wait on a counting semaphore.

```
int RTX_Wait_Sem ( int semID, long far *time_ms );
```

## Parameters

*semID*

The semaphore ID of a counting semaphore acquired by a call to **RTX_Create_Sem**.

*time_ms*

Pointer for ms to wait: Zero -> Wait forever for semaphore
Positive value -> Maximum number of milliseconds to wait for resource
Negative value -> Polling mode, return immediately. If semaphore was not available the return value will be the -27 time-out indication.

## Return Value

0 on success else **error** code.

## Comments

This call waits for a defined time on a semaphore and returns with an error if the semaphore is still in use by other callers ("busy" condition is represented by a semaphore count of zero). Otherwise the semaphore count is decremented by one and the caller is granted the semaphore.

The wait priority used here is 20. (See the **RTX_Wait_Sem_Priority** alternative API for explanation of wait priority.)

## See Also

**RTX_Get_Sem** **RTX_Signal_Sem** **RTX_Wait_Sem_Priority**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Find_Sem

Find a semaphore by name.

```
int RTX_Find_Sem ( int far *ID, const char far *name );
```

## Parameters

*ID*

Output parameter: Pointer to semaphore ID

*name*

Input parameter: Pointer to the four character name of the semaphore to search for.

## Return Value

zero -> Semaphore found
non-zero -> Return `error` code.

## Comments

The four character name string need not be zero terminated. However, it must contain four characters to be matched against the four character name provided when semaphore was created.

## See Also

RTX_Create_Sem

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Sleep_Fine

Go to sleep for a specified number of Real-Time Interrupts (RTI).

```
int RTX_Sleep_Fine ( unsigned int time_rti );
```

## Parameters

*time_rti*

Sleep time in RTI ticks

## Return Value

0 on success else error code.

## Comments

The RTX_Wakeup function can wake up a sleeping task before its sleep timer has expired.

If the task has an outstanding wake request pending when it calls here, the task will continue execution immediately without waiting.

## See Also

RTX_Sleep_Request RTX_Wakeup RTX_Sleep_Time
RTX_Sleep_Long

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| 1.20 | 1.20 | 1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Sleep_Long

Go to sleep for a large number of Real-Time Interrupts. (**RTI**).

```
int RTX_Sleep_Long ( long time_rti );
```

## Parameters

*time_rti*

31 bit sleep time in RTI ticks. This value is not valid if negative and the function will return immediately.

## Return Value

0 on success else **error** code.

## Comments

The **RTX_Wakeup** function can wake up a sleeping task before its sleep timer has expired.

If the task has an outstanding wake request pending when it calls here, the task will continue execution immediately without waiting.

## See Also

**RTX_Sleep_Request** **RTX_Wakeup** **RTX_Sleep_Time**
**RTX_Sleep_Fine**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| 1.20 | 1.20 | 1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Disable_Task_Scheduling

Disable task switching.

```
void RTX_Disable_Task_Scheduling ( void );
```

## Parameters

-- none --

## Return Value

-- none --

## Comments

This function locks the processor to the current task, preventing further task switching until either **RTX_Enable_Task_Scheduling** or a sleep API function is called. Entering any sleep API defeats the task lock. In this case the task lock condition is released until the task resumes execution. During the sleep, the system automatically releases the lock condition to allow other tasks to execute.

During the task lock condition, interrupts still execute provided that the user leaves interrupts enabled. Installed **Timer** procedures will not be executed.

This call must be followed by a call to **RTX_Enable_Task_Scheduling** as soon as possible to re-enable the task switching. If this period exceeds a hundred milliseconds then the watchdog should be triggered (using the Hardware API **hal_refresh_watchdog** function) by the user until the task switching is re-enabled.

Note that this is implemented as a spin lock, such that if for some reason the task calls here N times then N calls to **RTX_Enable_Task_Scheduling** are required to unwind the spin lock and re-enable the task switching.

## See Also

**RTX_Enable_Task_Scheduling** **RTX_Change_TaskPrio**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Enable_Task_Scheduling

Re-enable task switching.

```
void RTX_Enable_Task_Scheduling ( void );
```

## Parameters

-- none --

## Return Value

-- none --

## Comments

This function must be called following use of the **RTX_Disable_Task_Scheduling** function.

## See Also

**RTX_Disable_Task_Scheduling**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Get_TimeDate

Get system time and date.

```
int RTX_Get_TimeDate ( TimeDate_Structure far *td );
```

## Parameters

*td*

Output parameter, pointer to TimeDate_Structure **type** allocated by user and filled by this function.

## Return Value

0: Location at `td` contains system date and time

## See Also

**RTX_Set_TimeDate** **RTX_Get_System_Ticks** **RTX_Get_RTI_Ticks**
**RTX_GetFineTick** **RTX_GetTick_us** **RTX_GetTickCount**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Set_TimeDate

Set system time and date.

```
int RTX_Set_TimeDate ( const TimeDate_Structure far *td );
```

## Parameters

*td*

Input parameter, pointer to TimeDate_Structure **type** containing new time/date.

## Return Value

0: Success

## 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.

**Note:** This function performs no validity check on date/time data supplied by user.

## See Also

RTX_Get_TimeDate

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## RTX_Get_System_Ticks

Get tick count of system clock, in milliseconds.

```
void RTX_Get_System_Ticks ( unsigned long far *ticks );
```

## Parameters

*ticks*

Output parameter: System time in milliseconds.

## Return Value

-- none --

## Comments

This 32 bit up count rolls over to zero each 2**32 milliseconds, which is approximately each 1193 hours of continuous operation (= 41.71 days).

The alternate C-library API **RTX_GetTickCount** may provide a more comfortable access to the same value.

For real-time code, see discussion within the **RTX_Get_RTI_Ticks** documentation concerning use of the **_pRti_Tick** direct access pointer with in-line code.

## SC1x3/SC23 Comments

If the system's real-time interrupt has been **configured** for **rates** above 1000 Hz, the count here will differ from that returned by the **RTX_Get_RTI_Ticks** API.

## See Also

**RTX_Get_TimeDate** **RTX_Get_RTI_Ticks** **RTX_GetTickCount**
**RTX_GetFineTick** **RTX_GetTick_us**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_GetTickCount

Get tick count of system clock, in milliseconds.

```
unsigned long RTX_GetTickCount (void);
```

## Return Value

System time in milliseconds

## Comments

This 32 bit up count rolls over to zero each each 2**32 real-time interrupts, which is approximately each 1193 hours of continuous operation (= 41.71 days).

For real-time code, see discussion within the **RTX_Get_RTI_Ticks** documentation concerning use of the **_pRti_Tick** direct access pointer with in-line code.

## SC1x3/SC23 Comments

If the system's real-time interrupt has been **configured** for **rates** above 1000 Hz, the count here will differ from that returned by the **RTX_Get_RTI_Ticks** API.

## See Also

**RTX_Get_TimeDate** **RTX_Get_RTI_Ticks** **RTX_Get_System_Ticks**
**RTX_GetFineTick** **RTX_GetTick_us**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Get_RTI_Ticks

Get system's Real-Time Interrupt (RTI) count.

```
unsigned long RTX_Get_RTI_Ticks (void);
```

## Return Value

Real-Time interrupt count

## Comments

This 32 bit up count rolls over to zero each 2**32 real-time interrupts, which is approximately each 1193 hours of continuous operation (= 41.71 days) when operating at 1 kHz **RTI** rate.

A faster way to access the system's RTI up counter is to read the long word referenced by the **_pRti_Tick**. This alternate approach using inline code should be considered, especially when system time is required from within time-critical code such as Interrupt Service Routines. Keep in mind that you must momentarily mask interrupts around the 32 bit read from this location in order to assure consistent data. Also do not forget to call **RTX_FineTickInit** once at the start of your program so that this pointer will be initialized.

## SC1x Comments

For SC1x program's, this function is mapped to the **RTX_GetTickCount** API, which is equivalent for the fixed 1 kHz RTI rate. This allows common source code to be used for the two system types.

## See Also

**RTX_Get_TimeDate** **RTX_Get_System_Ticks** **RTX_Get_RTI_Rate**
**RTX_GetTickCount** **RTX_Get_System_Ticks** **_pRti_Tick**
**RTX_GetFineTick** **RTX_GetTick_us**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Fine_Timer

Install a high resolution timer callback procedure that will be periodically executed by the kernel.

```
int RTX_Fine_Timer (
             const TimerProc_Structure far *TProcPtr );
```

## Parameters

*TProcPtr*

Input parameter, pointer to a `TimerProc_Structure` **type**.

## Return Value

0 on success else **error** code.

## SC1x Comments

For SC1x systems which always operate at 1 kHz RTI rate, this API is mapped to the **RTX_Install_Timer** API by a macro defined in the library header file.

## SC1x3/SC23 Comments

A timer ID is output to the 16 bit location referenced by `timerID` **member** of your `TimerProc_Structure` .

Alternate method **RTX_Fine_TimerP** allows the timer installation to be performed based on direct parameters.

You must call either the **RTX_Start_Timer**, **RTX_Timer_Delay** or **RTX_Timer_Delay_RTI** 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 time consuming functions.

## See Also

**RTX_Fine_TimerP** **RTX_Start_Timer** **RTX_Remove_Timer**
**RTX_Install_Timer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Fine_TimerP

Install a high resolution timer callback procedure that will be periodically executed by the kernel.

```
int RTX_Fine_TimerP ( unsigned int far *timerID,
                      RTX_TIMER_CALLBACK proc,
                      unsigned long lParam,
                      unsigned long name,
                      long rti_ticks );
```

## Parameters

*timerID*

Output parameter, pointer to location to receive new timer ID.

*proc*

Far vector to timer callback procedure.

*lParam*

Any 32 bit value you would like to see again on each entry into the timer procedure.

*name*

Provisional 4 character timer name (little endian order), not of any use in current implementation.

*rti_ticks*

Interval between calls to timer procedure in Real-Time Interrupt (**RTI**) ticks.

## Return Value

0 on success else **error** code.

## SC1x Comments

For SC1x systems which always operate at 1 kHz RTI rate, this API is mapped to the **RTX_Install_TimerP** API by a macro defined in the library header file.

## SC1x3/SC23 Comments

The RTX_TIMER_CALLBACK prototype has the form:

```
void huge RTX_TIMER_CALLBACK(
      unsigned int timer_id, unsigned long lParam )
```

The timer callback procedure's input parameters may be ignored and the function declared with void parameters. The `timer_id` parameter is a copy of the timer ID output by this function. The `lParam` input parameter is a copy of the `lParam` parameter you supplied to `RTX_Fine_TimerP` .

You must call either the **RTX_Start_Timer**, **RTX_Timer_Delay** or **RTX_Timer_Delay_RTI** 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 time consuming functions.

## See Also

**RTX_Fine_Timer** **RTX_Start_Timer** **RTX_Remove_Timer**
**RTX_Install_TimerP**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Install_Timer

Install a timer callback procedure that will be periodically executed by the kernel.

```
int RTX_Install_Timer (
             const TimerProc_Structure far *TProcPtr );
```

## Parameters

*TProcPtr*

Input parameter, pointer to a `TimerProc_Structure` **type**.

## Return Value

0 on success else **error** code.

## Comments

A timer ID is output to the 16 bit location referenced by `timerID` **member** of your `TimerProc_Structure` .

Alternate method **RTX_Install_TimerP** allows the timer installation to be performed based on direct parameters.

You must call either the **RTX_Start_Timer**, **RTX_Timer_Delay** or **RTX_Timer_Delay_RTI** 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 time consuming functions.

## See Also

**RTX_Install_TimerP** **RTX_Start_Timer** **RTX_Remove_Timer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Install_TimerP

Install a timer callback procedure that will be periodically executed by the kernel.

```
int RTX_Install_TimerP ( unsigned int far *timerID,
                         RTX_TIMER_CALLBACK proc,
                         unsigned long lParam,
                         unsigned long name,
                         long interval );
```

## Parameters

*timerID*

Output parameter, pointer to location to receive new timer ID.

*proc*

Far vector to timer callback procedure.

*lParam*

Any 32 bit value you would like to see again on each entry into the timer procedure.

*name*

Provisional 4 character timer name (little endian order), not of any use in current implementation.

*interval*

Milliseconds interval between calls to timer procedure.

## Return Value

0 on success else **error** code.

## Comments

The RTX_TIMER_CALLBACK prototype has the form:

```
void huge RTX_TIMER_CALLBACK(
      unsigned int timer_id, unsigned long lParam )
```

The timer callback procedure's input parameters may be ignored and the function declared with void parameters. The `timer_id` parameter is a copy of the timer ID output by this function. The `lParam` input parameter is a copy of the `lParam` parameter you supplied to `RTX_Install_TimerP` .

You must call either the **RTX_Start_Timer**, **RTX_Timer_Delay** or **RTX_Timer_Delay_RTI** 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 time consuming functions.

## See Also

**RTX_Install_Timer** **RTX_Start_Timer** **RTX_Remove_Timer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Remove_Timer

Stop execution and remove a timer procedure.

```
int RTX_Remove_Timer ( unsigned int timerID );
```

## Parameters

*timerID*

The timer ID output from **RTX_Install_Timer**.

## Return Value

0 on success else **error** code.

## See Also

**RTX_Install_Timer** **RTX_Start_Timer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Start_Timer

Start periodic execution of timer procedure.

```
int RTX_Start_Timer ( unsigned int timerID );
```

## Parameters

*timerID*

The timer ID output from **RTX_Install_Timer**.

## Return Value

0 on success else **error** code (invalid timer ID).

## See Also

**RTX_Timer_Delay** **RTX_Install_Timer** **RTX_Stop_Timer**
**RTX_Timer_Delay_RTI**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Stop_Timer

Stop execution of timer procedure.

```
int RTX_Stop_Timer ( unsigned int timerID );
```

## Parameters

*timerID*

The timer ID output from **RTX_Install_Timer**.

## Return Value

0 on success else **error** code.

## See Also

**RTX_Timer_Delay** **RTX_Install_Timer** **RTX_Start_Timer**
**RTX_Timer_Delay_RTI**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Timer_Delay

Set timer delay interval in milliseconds.

```
int RTX_Timer_Delay ( unsigned int timerID
                      long delay_ms);
```

## Parameters

*timerID*

The timer ID output from **RTX_Install_Timer**.

*delay_ms*

Delay period in milliseconds. If this value is zero the timer is deactivated.

## Return Value

0 on success else **error** code.

## Comments

This function can be used on periodic timers to set the interval to the next call, thereby providing a phase control.

For non-periodic timers this function can be called with varying delay values.

When 'delay_ms' is set to '1', this function is identical to the **RTX_Start_Timer** function (assuming 1 kHz RTI operation).

When 'delay_ms' is set to '0', this function is identical to the **RTX_Stop_Timer** function.

## See Also

**RTX_Install_Timer** **RTX_Start_Timer** **RTX_Stop_Timer**
**RTX_Timer_Delay_RTI**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Timer_Delay_RTI

Set timer delay interval in units of Real-Time Interrupt (RTI) ticks.

```
int RTX_Timer_Delay_RTI ( unsigned int timerID
                          long rti_ticks);
```

## Parameters

*timerID*

The timer ID output from **RTX_Install_Timer**.

*rti_ticks*

Number of **RTI** ticks for timer delay period. If this value is zero the timer is deactivated.

## Return Value

0 on success else **error** code.

## SC1x Comments

For SC1x systems this API is identical to the **RTX_Timer_Delay** API.

## SC1x3/SC23 Comments

This function can be used on periodic timers to set the interval to the next call, thereby providing a phase control.

For non-periodic timers this function can be called with varying delay values.

When 'rti_ticks' is set to '1', this function is identical to the **RTX_Start_Timer** function.

When 'rti_ticks' is set to '0', this function is identical to the **RTX_Stop_Timer** function.

## See Also

**RTX_Install_Timer** **RTX_Start_Timer** **RTX_Stop_Timer**
**RTX_Timer_Delay**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Install_RTI_Hook

This function installs a callback function which executes periodically within the 1000 Hz (default rate) Real-Time Interrupt (RTI).

```
void RTX_Install_RTI_Hook (
                RTI_HOOK vector,
                unsigned char delay   )
```

## Parameters

*vector*

Far pointer to callback procedure to be invoked from within 1000 Hz RTI. Set this argument to NULL to remove a callback.

```
typedef unsigned char huge (far * RTI_HOOK ) (void) ;
```

*delay*

Number of 1000 Hz RTI before first call. Set to 1 for callback on very next RTI. Set to 0 for 256 RTI later.

## Return Value

-- none --

## Comments

External interrupts are masked and then restored to their state on entry.

The term 'tick' used here refers to a 1 millisecond period when the system timer RTI is operated at its default rate of 1000 Hz. Note that this RTI rate can be configured to higher rates with the **[TIMER] RTI** setting in the CHIP.INI, in which case tick time will be under 1 millisecond.

The callback function is free to use any register except for the DS and DI registers. **These two registers must be preserved.**

This API was provided for applications which require a precise and variable timing for high frequency front end hardware sampling, for example.

The least significant byte of the callback's return value (AL register) will be used as the delay count in milliseconds to the next callback. Examples (assuming 1000Hz RTI):

- Return 1 to execute again on very next RTI, 1 millisecond later.
- Return zero for a delay of 256 milliseconds.

- Return 5 to execute at a 200 Hz rate.

The system supports only a single RTI hook. Installing a second overwrites the first. A callback can be removed by installing a NULL callback vector.

Unlike with the RTX **timer** callback functions, which execute after a task switch to the kernel task, this RTI hook executes directly in the context of the interrupt. No task switch occurs prior to the callback and execution proceeds on the stack of what ever task was interrupted. Consequently the callback should be conservative with its stack space usage.

This interrupt has been registered with the RTX kernel, so the user is free to use the task triggering RTX API functions from within the callback such as:

**RTX_Wakeup_Task**
**RTX_Resume_Task**
**RTX_Signal_Sem**
**RTX_Signal_Events**
**RTX_Send_Msg**

However any time consuming activity must be avoided and it is not legal to call any RTOS API which could block or result in current task being suspended. As a rough guideline, the dwell within this callback should be held to under 50 us. Either very carefully written C code, or assembly code is recommended here.

As true for any installed callback function, it is essential that an application program remove its RTI hook callback (`vector` set to zero) before program termination.

## See Also

**RTX_Install_Timer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Create_EventGroup

Create an event group.

```
int RTX_Create_EventGroup ( unsigned int far *id_ptr,
                            const char far *name,
                            unsigned int init_value );
```

## Parameters

*id_ptr*

Output parameter, pointer to where event group ID will be put by this function if successful.

*name*

Input parameter, 4 character name for new event group (need not be zero terminated, but must be 4 bytes).

*init_value*

Initial value of the 16 event flags of the group.

## Return Value

0 on success else **error** code (no free event group available).

## Comments

Each event group includes 16 event flags. Each event group has a unique ID. You can provide a unique 4 character tag to identify the event group, for use with the **RTX_Find_EventGroup** function.

## See Also

**RTX_Delete_EventGroup** **RTX_Find_EventGroup** **RTX_Wait_For_Event**
**RTX_Signal_Events** **RTX_Get_EventGroup_State** **RTX_Get_Saved_Events**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Delete_EventGroup

Delete an event group.

```
int RTX_Delete_EventGroup ( int id );
```

## Parameters

*id*

Event group ID produced by **RTX_Create_EventGroup** call.

## Return Value

0 on success else **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.

## See Also

**RTX_Create_EventGroup**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## RTX_Signal_Events

Signal that one or more events in a group have occurred by setting the event flags. If any tasks are waiting on these events, the Event Manager will now wake them up.

```
int RTX_Signal_Events ( unsigned int id,
                        unsigned int event_mask,
                        unsigned int event_value );
```

## Parameters

*id*

Event group ID produced by **RTX_Create_EventGroup** call.

*event_mask*

16-Bit mask identifying the flags of interest in the group

*event_value*

Event value for the 16 event flag/bits. Only the bits marked '1' in the event_mask are relevant here.

## Return Value

0 on success else **error** code.

## Comments

This function modifies the event group bits marked by ones in the `event_mask` to either '1' or '0' based on the values of the corresponding bits in the `event_value` parameter.

Event group bits which are zero in the `event_mask` are unaffected.

Note that "event occurred" condition can be either a bit '1' or '0'. This is application defined and is specified by the `value` supplied to the **RTX_Wait_For_Event** call.

## See Also

**RTX_Create_EventGroup** **RTX_Wait_For_Event** **RTX_Get_EventGroup_State** **RTX_Wait_For_EventP**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Wait_For_Event

The calling task waits until the Event Manager wakes it up (the event has occurred) or the specified timeout is reached.

```
int RTX_Wait_For_Event ( unsigned int id,
                         RTX_Wait_Event far *event_ptr );
```

## Parameters

*id*

Event group ID produced by **RTX_Create_EventGroup** call.

*event_ptr*

Pointer to RTX_Wait_Event **structure** filled in by caller.

## Return Value

0 on success else **error** code.

## Comments

Note that the Event Manager's wakeup mechanism is level sensitive (not edge triggered). Consequently, a task may want to acknowledge an event bit level after a wakeup by negating the flag using the **RTX_Signal_Events** function following a wakeup.

The alternative **RTX_Wait_For_EventP** function is identical to this function, except for parameter passing method.

The **RTX_Get_Saved_Events** function can be used to pick up the event states.

## See Also

**RTX_Create_EventGroup** **RTX_Signal_Events** **RTX_Get_EventGroup_State** **RTX_Wait_For_EventP**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Wait_For_EventP

The calling task waits until the Event Manager wakes it up (the event has occurred) or the specified timeout is reached.

```
int RTX_Wait_For_EventP ( unsigned int id, unsigned int mask,
                          unsigned int value, int match,
                          long timeout );
```

## Parameters

*id*

Event group ID produced by **RTX_Create_EventGroup** call.

*mask*

16-Bit mask identifying the flags of interest in the group

*value*

Event value for the 16 event flag/bits. Only the bits marked '1' in the mask are relevant here.

*match*

=0 (OR): only one flag must match with value
!=0 (AND): all flags must match with value

*timeout*

Positive value -> wait up to this number of milliseconds for an event match
Zero -> wait forever for an event match
Negative value -> No wait, return immediately

## Return Value

0 on success else **error** code.

## Comments

Note that the Event Manager's wakeup mechanism is level sensitive (not edge triggered). Consequently, a task may want to acknowledge an event bit level after a wakeup by negating the flag using the **RTX_Signal_Events** function following a wakeup.

The function is identical to **RTX_Wait_For_Event**, except for manner of parameter passing.

The **RTX_Get_Saved_Events** function can be used to pick up the event states.

## See Also

**RTX_Create_EventGroup** **RTX_Signal_Events** **RTX_Get_EventGroup_State**
**RTX_Wait_For_Event**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Get_EventGroup_State

Read current events on a group.

```
int RTX_Get_EventGroup_State ( unsigned int id,
                               unsigned int far *event );
```

## Parameters

*id*

Event group ID produced by **RTX_Create_EventGroup** call.

*event*

Output parameter, pointer to 16 bit location to receive the current state of the event flags.

## Return Value

0 on success else **error** code (invalid group ID).

## Comments

This function may be used by any program thread to acess the event group's current flag values.

## See Also

**RTX_Create_EventGroup** **RTX_Signal_Events** **RTX_Get_Saved_Events**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Get_Saved_Events

Return the state of the 16 event flags as they were at the time the calling task most recently completed a `RTX_Wait_For_Event` (or `RTX_Wait_For_EventP` ) call.

```
int RTX_Get_Saved_Events ( unsigned int far *Event );
```

## Parameters

*`Event`*

Output parameter: Saved events.

## Return Value

0: Success.

## Comments

This function accesses the event flags stored for this specific program thread. The event group was specified at the time of the `RTX_Wait_For_Event` call.

Use the **RTX_Get_EventGroup_State** function to read current event group flag states.

## See Also

**RTX_Create_EventGroup** **RTX_Signal_Events** **RTX_Wait_For_Event**
**RTX_Wait_For_EventP** **RTX_Get_EventGroup_State**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Find_EventGroup

Find an event group by specified name tag and return the unique event group ID.

```
int RTX_Find_EventGroup ( unsigned int far *id_ptr,
                          const char far *name );
```

## Parameters

*id_ptr*

Output parameter, pointer to where to put the event group ID if found.

*name*

Input parameter, pointer to 4 character name of event group to look for.

This name does not need to be zero terminated, but must contain 4 bytes.

## Return Value

0 on success else **error** code.

## Comments

If more than one event group exists with the specified name, this function will return the ID of one of these event groups. Which event group is returned is indeterminant when the name is not unique.

## See Also

**RTX_Create_EventGroup**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Create_Msg

A message exchange must be created by an application before it can be used. The Message Exchange Manager outputs a 16-bit unique ID to the caller from this call. You can provide a unique 4 character tag to identify the message exchange.

```
int RTX_Create_Msg ( RTX_Msg far *msg_ptr );
```

## Parameters

*msg_ptr*

Input/Output parameter, pointer to a RTX_Msg **structure**.

## Return Value

0 on success else **error** code.

## Comments

The caller must to fill in portions of the RTX_Msg structure prior to calling. The new message exchange's ID (handle) is returned in this structure.

## See Also

**RTX_Delete_Msg** **RTX_Find_Msg** **RTX_Send_Msg**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## RTX_Delete_Msg

Delete a message exchange.

```
int RTX_Delete_Msg ( unsigned int id );
```

## Parameters

*id*

The message exchange ID output on call to **RTX_Create_Msg**.

## Return Value

0 on success else **error** code message exchange still in use or invalid ID).

## See Also

**RTX_Create_Msg**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## RTX_Send_Msg

Send provided message to a specified message exchange.

```
int RTX_Send_Msg ( unsigned int id, int prio,
                   const void far *msg );
```

## Parameters

*id*

The message exchange ID output on call to **RTX_Create_Msg**.

*prio*

Message priority (mailbox) 0 - 3 where 0 is highest priority

*msg*

Pointer to a 12 byte message to be sent

## Return Value

0 on success else **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. A task switch occurs immediately if called for based on task priorities.

The format of the 12 byte message being sent is defined by the application program.

Messages will be reported in message priority order, and from each priority queue in FIFO order.

## See Also

**RTX_Get_Msg** **RTX_Wait_For_Msg** **RTX_Wait_For_MsgP**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS C Library - RTOS API

---

## RTX_Get_Msg

Get message without waiting (polling) from a specified message exchange.

```
int RTX_Get_Msg ( unsigned int id, void far *msg );
```

## Parameters

*id*

The message exchange ID output on call to **RTX_Create_Msg**.

*msg*

Output parameter: Pointer to a 12 byte user buffer for storing the message (if any).

## Return Value

0 on success else **error** code.

## Comments

This function returns immediately with return value -28 if no message is available.

The format of the 12 byte message being sent is defined by the application program.

When more than one message is available at the message exchange, the oldest (FIFO order) message from the highest priority message queue will be reported.

## See Also

**RTX_Send_Msg** **RTX_Wait_For_Msg** **RTX_Wait_For_MsgP**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Wait_For_Msg

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.

```
int RTX_Wait_For_Msg ( RTX_Wait_Msg far *msg_ptr );
```

## Parameters

*msg_ptr*

Input parameter: pointer to a `RTX_Wait_Msg` **structure** containing parameters.

## Return Value

0 on success else **error** code, invalid ID or -28: no message available.

## Comments

This procedure is identical to **RTX_Wait_For_MsgP**, except for method of parameter passing.

The format of the 12 byte message received is defined by the application program.

When more than one message is available at the message exchange, the oldest (FIFO order) message from the highest priority message queue will be reported. Note that message priority and the wait priority,

`msg_ptr->prio` ,

are independent and unrelated concepts.

## See Also

**RTX_Send_Msg** **RTX_Get_Msg** **RTX_Wait_For_MsgP**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Wait_For_MsgP

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.

```
int RTX_Wait_For_MsgP ( unsigned int msgID, unsigned int prio,
                        void far *msg, long timeout );
```

## Parameters

*msgID*

The message exchange ID output on call to **RTX_Create_Msg**.

*msg*

Pointer to a 12 byte user buffer for storing the message (if any)

*timeout*

Maximum time (milliseconds) to wait for a message:
timeout == 0 -> Wait forever
timeout < 0 -> invalid (must be >= 0)

*prio*

priority for wait (0-0xFFFF), 0=highest. Specify here the priority of the calling task's access to the messages. To wait in FIFO order, have all 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.

## Return Value

0 on success else **error** code, invalid ID or -28: no message available.

## Comments

This procedure is identical to **RTX_Wait_For_Msg**, except for method of parameter passing.

The format of the 12 byte message received is defined by the application program.

When more than one message is available at the message exchange, the oldest (FIFO order) message from the highest priority message queue will be reported. Note that message priority and the wait priority,

`msg_ptr->prio`,

are independent and unrelated concepts.

## See Also

**RTX_Send_Msg** **RTX_Get_Msg** **RTX_Wait_For_Msg**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_Find_Msg

Find a message exchange, specified by 4 character name tag.

```
int RTX_Find_Msg ( unsigned int far *id_ptr,
                   const char far *name);
```

## Parameters

*id_ptr*

Output parameter, pointer to location in callers memory which will receive the Message exchange ID if a message exchange with specified name is found.

*name*

Input parameter, pointer to 4 character name of message exchange to look for. (This string need not be zero terminated, but must be four bytes.)

## Return Value

0 on success else **error** code.

## 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.

## See Also

**RTX_Create_Msg**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

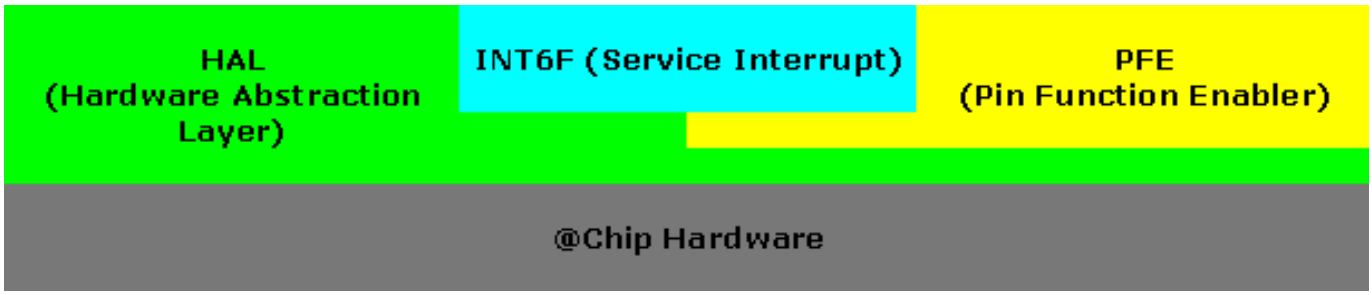
# @CHIP-RTOS - Hardware API Layers



## Hardware API Layers

Hardware API

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 @CHIP-RTOS. 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 minimizes hardware dependencies.

End of document




# @CHIP-RTOS - I2C Bus and SPI Interface



## I2C / SPI API

Here are the interface definitions for access to the IPC@CHIP®'s I2C bus and SPI bus.

Philips was the inventor of the Inter-IC or I²C-bus, and it is now firmly established as the worldwide de-facto solution for embedded applications. It is used extensively in a variety of microcontroller-based professional, consumer and telecommunications applications as a control, diagnostic and power management bus. As a two-wire serial bus, its inherently simple operation was crucial to its emergence as the worldwide de-facto standard.

SPI is a serial bus standard established by Motorola. The Serial Peripheral Interface (SPI) is a synchronous serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be Serial EEPROMs, shift registers, display drivers, A/D converters, etc.

### Interrupt 0xAA Services

The I2C Bus / SPI 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 and SPI Bus of the IPC@CHIP® for application programs.

### General

- **0x00: Install dynamic link, then call function**

### I2C Functions

- **0x80: I2C Master: Initialize the I2C Bus**
- **0x81: I2C Master: Scan I2C Devices**
- **0x85: I2C Master: Scan I2C Devices (extended address)**
- **0x82: I2C Master: Transmit / Receive Character**
- **0x86: I2C Master: Transmit / Receive Character (extended address)**
- **0x83: I2C Master: Transmit / Receive Block**
- **0x87: I2C Master: Transmit / Receive Block (extended address)**
- **0x84: I2C Master: Release I2C Bus**
- **0x8B: I2C Master: Restart the I2C Bus**
- **0x8E: I2C Master: Select I2C Clock Pin**
- **0x8F: I2C Master: Select I2C Data Pin**
- **0x88: I2C Master: Set the I2C Bus timeout**
- **0xB0: I2C Master/Slave: Set the I2C Bus speed**
- **0xB1: I2C Master/Slave: Reset the I2C Bus controller**

- 0xB2: I2C Slave: Set I2C Bus slave address
- 0xB3: I2C Slave: Read I2C Bus state
- 0xB4: I2C Slave: Send acknowledge on I2C bus
- 0xB5: I2C Slave: Read byte from I2C bus
- 0xB6: I2C Slave: Write byte to I2C bus
- 0xB7: I2C Slave: Return to idle state

## SPI Functions

- 0x90: SPI Master: Initialize the master interface
- 0x95: SPI Master: Write Block
- 0x96: SPI Master: Read Block
- 0x97: SPI Master: Read and Write Block
- 0xA0: SPI Slave: Initialize the slave interface
- 0xA5: SPI Slave: Write
- 0xA6: SPI Slave: Read
- 0xA7: SPI Slave: Read and Write
- 0xA8: SPI Master: Wait on / Release semaphore

---

## Interrupt 0xAA service 0x80: I2C Master: Initialize the I2C Bus

This function initializes the I2C bus. It enables the two defined pins for I2C usage (see also function 8eh and 8fh).

## Parameters

*AH*
0x80

## Return Value

AX=0, Carry flag cleared

## 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.

## SC1x3/SC23 Comments

On SC1x3/SC23 the default is to use the hardware I2C interface on the dedicated pins. If the user calls functions 8eh and 8fh to select other PIOs for the bus, the I2C bus will be software emulated on the selected PIO pins.

If the hardware interface is used, the calling task will go to sleep, until the I2C interrupt issues a wakeup. Therefore the user should not make own wakeup calls to the I2C API calling task, while sleeping inside I2C functions.

## Related Topics

Select I2C Clock Pin

Select I2C **Data** Pin
PFE: **Enable** Programmable I/O Pins

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0x81: I2C Master: Scan I2C Devices

Report addresses of slave devices, one at a time.

## Parameters

*AH*
0x81

*AL*
Bit7-0: First slave address (even address, LSB=0)

*CL*
Bit7-0: 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.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0x85: I2C Master: Scan I2C Devices (extended address)

Report addresses of slave devices, one at a time.

## Parameters

*AH*

0x85

*DX*

Bit10-0: First slave address (even address, LSB=0)

*CX*

Bit10-0: Last slave address (even address, LSB=0)

## Return Value

AX: 0 no slave found
AX: -1 Timeout
AX: 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.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0x82: I2C Master: Transmit / Receive Character

Send or receive a single character.

## Parameters

*AH*

0x82

*AL*

Bit7-0: 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

If the direction or the slave address changes, this functions executes the I2C restart function automatically. This will insert a I2C start condition on the bus. After this, the I2C address is send again.

**I2C error codes:**

8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Comments

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read (change from master to slave)
5: Arbitration lost, own slave address received, master wants to write (change from master to slave)
6: Arbitration lost
7: Bus error

## Related Topics

I2C Transmit/Receive **Block**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0x86: I2C Master: Transmit / Receive Character (extended address)

Send or receive a single character.

## Parameters

*AH*
0x86

*DX*
Bit10-0: 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

If the direction or the slave address changes, this functions executes the I2C restart function automatically. This will insert a I2C start condition on the bus. After this, the I2C address is send again.

**I2C error codes:**

8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Comments

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read (change from master to slave)
5: Arbitration lost, own slave address received, master wants to write (change from master to slave)
6: Arbitration lost
7: Bus error
8: Timeout

## Related Topics

I2C Transmit/Receive **Block**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0x83: I2C Master: Transmit / Receive Block

## Parameters

*AH*
0x83

*AL*
Bit7-0: Slave address, LSB:0 => Transmit, LSB:1 => Receive

*CX*
Number of bytes to transmit or receive

*ES:BX*
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.

If the direction or the slave address changes, this functions executes the I2C restart function automatically. This will insert a I2C start condition on the bus. After this, the I2C address is send again.

## Related Topics

I2C Transmit/Receive **Character**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0x87: I2C Master: Transmit / Receive Block (extended address)

## Parameters

*AH*
0x87

*DX*
Bit10-0: Slave address, LSB:0 => Transmit, LSB:1 => Receive

*CX*
Number of bytes to transmit or receive

*ES:BX*
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.

If the direction or the slave address changes, this functions executes the I2C restart function automatically. This will insert a I2C start condition on the bus. After this, the I2C address is send again.

## Related Topics

I2C Transmit/Receive **Character**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0x84: I2C Master: Release I2C Bus

## Parameters

*AH*

0x84

## Return Value

Carry flag cleared

## Comments

This function generates an I2C stop condition. (The Bus will be released). At next call of transmit or receive functions, the address will be send again.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0x8B: I2C Master: Restart the I2C Bus

## Parameters

*AH*
0x8B

## Return Value

Carry flag cleared

CF: 0

## Comments

This function generates an I2C start condition. (The Bus will be reserved)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0x8E: I2C Master: Select I2C Clock Pin

Select IPC@CHIP® I/O pin to be used for I2CCLK signal.

## Parameters

*AH*
0x8E

*AL*
PIO pin number

## Return Value

none

## SC1x 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.

## SC1x3/SC23 Comments

To use the I2C hardware interface do not call the functions "Select I2C Clock Pin" and "Select I2C Data Pin" or call them both with PIO pin number 0.

## Related Topics

**Initialize** I2C Bus Function
Select I2C **Data** Pin

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0x8F: I2C Master: Select I2C Data Pin

Select IPC@CHIP® I/O pin to be used for I2CDAT signal.

## Parameters

*AH*
0x8F

*AL*
PIO pin number

## Return Value

Carry flag cleared

## SC1x 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.

## SC1x3/SC23 Comments

To use the I2C hardware interface do not call the functions "Select I2C Clock Pin" and "Select I2C Data Pin" or call them both with PIO pin number 0.

## Related Topics

**Initialize** I2C Bus Function
Select I2C **Clock** Pin

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0x88: I2C Master: Set the I2C Bus timeout

This function sets the I2C bus timeout. If you like to change the default timeout value, then call this function after the I2C init function.

## Parameters

*AH*

0x88

*BX*

Timeout value.

If the I2C hardware interface is used, this value has a resolution in milliseconds. This means that the internal I2C functions will wait for BX milliseconds if the I2C clock signal is released by the I2C slave.

If the software I2C emulation is used, this value is interpreted as a cycle count. This means that the internal I2C functions check BX times if the I2C clock signal is released by the I2C slave.

## Return Value

Carry flag cleared

## Comments

If the I2C slave uses clock stretching, this timeout specifies the maximum waiting time, before the access is aborted.

The default timeout is:
SC12: 50 cycles
SC12B: 100 cycles
SC13/SC11: 250 cycles
SC1x3/SC23: 3000 cycles / 50 ms

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0xB0: I2C Master/Slave: Set the I2C Bus speed

This function sets the I2C bus speed, if the I2C hardware interface is used.

## Parameters

*AH*

0xB0

*BL*

N (allowed value range 0-7)

*BH*

M (allowed value range 0-15)

## Return Value

AX=0, Carry flag cleared

## Comments

FREQsample = CPU clock / 2^N
FREQoscl = CPU clock / ((2^N) * (M+1) * 10)

FREQoscl is the frequency of the I²C clock line when the I²C controller is in master mode.
FREQsample is the frequency at which the I²C bus is sampled. This frequency should be at least 10 times faster than the fastest master on the bus, to ensure that START and STOP conditions are always detected.

Default is N=4, M=5, which sets FREQsample to 6 MHz and FREQoscl to 100 KHz at 96 MHz CPU frequency.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0xB1: I2C Master/Slave: Reset the I2C Bus controller

This function resets the I2C bus hardware controller.

## Parameters

*AH*
0xB1

## Return Value

Carry flag cleared

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0xB2: I2C Slave: Set I2C Bus slave address

This function initializes the I2C bus slave interface. To use the I2C slave interface the user has to install an interrupt service routine for the I2C interrupt. An interrupt occurs when the I2C state changes. The user should than read the I2C state with function **read I2C state.**

## Parameters

*AH*
0xB2

*BX*
Slave address:
Bit0: read/write bit must be 0
Bit7-1: 7 bit slave address or
Bit10-1: 10 bit slave address

*CL*
0: don't listen to general call address
1: listen to general call address

## Return Value

Carry flag cleared

## Comments

The I2C bus intialize function must be called previously. The Slave mode works only if the I2C hardware interface in used.

## Related Topics

Init I2C function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0xB3: I2C Slave: Read I2C Bus state

This function reads the current I2C bus state, if the I2C slave interface is used.

## Parameters

*AH*
0xB3

## Return Value

AX contains I2C bus state:
3: General call address received
4: Own slave address received, master wants to read
5: Own slave address received, master wants to write
6: Arbitration lost
7: Bus error

10: Data byte received from master
11: Data byte transmitted to master, master wants next byte
12: Last data byte transmitted to master
13: STOP or repeated START condition received

## Comments

This function should be used inside the I2C interrupt handler, if the I2C slave interface is used.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0xB4: I2C Slave: Send acknowledge on I2C bus

This function sends an acknowledge, if the I2C slave interface is used.

## Parameters

*AH*
0xB4

*AL*
0: Send a non acknowledge
1: Send acknowledge

## Return Value

Carry flag is cleared.

## Comments

This function should be used inside the I2C interrupt handler, if the I2C slave interface is used and the return state signals that the master wants to write a byte.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0xB5: I2C Slave: Read byte from I2C bus

This function reads a byte from the bus, if the I2C slave interface is used.

## Parameters

*AH*
0xB5

## Return Value

AL contains read byte, Carry flag is cleared.

## Comments

This function should be used inside the I2C interrupt handler, if the I2C slave interface is used and the return state signals that the master has written a byte, which can now be read out.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0xB6: I2C Slave: Write byte to I2C bus

This function writes a byte to the bus, if the I2C slave interface is used.

## Parameters

*AH*
0xB6

*AL*
Byte to write

*BL*
1: Signal that more bytes are available
0: Signal that this is the last byte that can be transfered

## Return Value

Carry flag is cleared.

## Comments

This function should be used inside the I2C interrupt handler, if the I2C slave interface is used and the return state signals that the master wants to read a byte.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAA service 0xB7: I2C Slave: Return to idle state

This function sets the I2C internal state back to idle, if the I2C slave interface is used.

### Parameters

*AH*
0xB7

### Return Value

Carry flag is cleared.

### Comments

This function should be used inside the I2C interrupt handler, if the I2C slave interface is used and the return state signals that the last byte was transmitted to the master.

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

## Interrupt 0xAA service 0x00: Install dynamic link, then call function

This function installs a far JMP opcode in the vicinity of the return instruction pointer to enter the specified USB API function directly from the user code. This function is provided for efficiency in real-time applications.

### Parameters

*AH*
0x00

*AL*
Number of I2C/SPI API function that is to be linked

### Return Value

Instruction Pointer is moved back 10 bytes so that newly installed far JMP will execute immediately.

(Refer to respective linked function's documention for further return value information.)

Note: All dynamically linked functions are free to use all registers except for DS, SI, DI, and BP registers which are preserved. This matches Paradigm C, Borland C++ 5.02 and Microsoft Visual C 1.52 calling conventions.

## Comments

Dynamic linking is not yet implemented for I²C!

A JMP FAR instruction will be written 10 bytes before the user's return instruction pointer. This far jump will transfer control directly into the RTOS from the user code when executed. This new far JMP will execute immediately upon return from this software interrupt due to the instruction pointer adjustment performed within the call.

Interrupts are masked and restored to their state on entry.

Since the code at calling site will be patched by this RTOS call, it is important that it has exactly the following form (the link index carried in AL register will vary)

```
JMP SHORT over_patch  ; Occupies 2 bytes of instruction code here
NOP    ; Never executed (1 byte)
NOP    ; Never executed (1 byte)
NOP    ; never executed (1 byte)
over_patch:  ; Linker overwrites above 5 bytes with JMP FAR instruction
MOV AX,00090h  ; AH = 0x00, AL = 0x90 = index for spi_init() or spi_init_hw()
INT 0AAh
```

## Developer Notes

This API service is intended to be used by the RTOS C-Library exclusively.
It should not be used within user applications.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0x90: SPI Master: Initialize the master interface

Initialize the SPI master interface.

SPI specifies four signals:

- clock (SCLK)
- master data output, slave data input (MOSI)
- master data input, slave data output (MISO)
- slave select (CSS)

In software SPI mode, the slave select signal pin is not specified with the SPI init function and not activated with the SPI read and write functions. The user has to enable any other PIO pin for that purpose and should serve this pin manually.

**SC1x3/SC23:**
In hardware SPI mode, if Autodrive-mode is enabled the SLVSEL pin is automatically driven by the SPI master read and write functions.

## Parameters

*AH*

Must be 0x90

*CL*

PIO number for SPI Clock Pin (SCLK)

*CH*

PIO number for SPI Input Pin (MISO)

*DL*

PIO number for SPI Output Pin (MOSI)

## SC1x Parameters

*BX*

Mode, must be 0 (SPI mode 0)

## SC1x3/SC23 Parameters

*BX*

Mode when software SPI is selected:
BX must be 0

Mode when hardware SPI is selected:
BIT0-1 = SPI mode (0-3)
BIT2 = 1 use Open-Drain-Mode, 0 use Output-Mode
BIT3 = 1 Autodrive SLVSEL chipselect pin, 0 do not drive SLVSEL pin
(Autodrive works only together with Alternate I/O)
BIT4 = 1 invert SLVSEL chipselect pin (active high), 0 do not invert SLVSEL pin (active low)
BIT5-7 = bit shift count, 000 = 8 bit, 001 = 1 bit ... 111 = 7 bit
BIT8 = 0 Normal I/O pinout, 1 Alternate I/O pinout (swap SDI and SDO pin)
BIT9-15 = must be 0

*SI*

SPI speed clock divider (valid values: 0-65535) (valid for hardware SPI only)
The SPI clock frequency is calculated with the formula:
SPI freq = CPU freq / (divider*2+2)

## Return Value

none

## SC1x3/SC23 Comments

If CL=CH=DL=0 than Hardware SPI is selected on dedicated pins else software SPI is performed on selected PIO pins.

A Wired-OR mode is provided which allows multiple masters to collide on the bus without risk of damage. In this mode, an external pull-up resistor is required on the Master Out Slave In (MOSI) and Master In Slave Out (MISO) pins. Generally, pull-ups are recommended on all of the external SPI signals to ensure they are held in a valid state, even when the SPI interface is disabled.

Alternate I/O pinout means that TX and RX lines are swapped.

## Related Topics

**Read SPI Block (master interface)**
**Write SPI Block (master interface)**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0x95: SPI Master: Write Block

Write n bytes out to the SPI Output pin

## Parameters

*AH*
Must be 0x95.

*ES:SI*
Pointer to write buffer

*CX*
Number of bytes in buffer to write

## Return Value

Result code in AX:
- Always 0 for software SPI
- 1, if illegal arguments were passed
- Else status bitmap:
  - BIT5 = 1: A collision has occurred between multiple masters
  - BIT6 = 1: An overrun was detected

## SC1x3/SC23 Comments

Overrun detection only works with hardware SPI and if the SLVSEL pin is used as chipselect.

## Related Topics

Init SPI (master interface)
Read Block SPI (master interface)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0x96: SPI Master: Read Block

Reads n bytes from the SPI Input pin

## Parameters

*AH*
Must be 0x96

*ES:DI*
Pointer to buffer for storing read data

*CX*
Number of bytes to read

## Return Value

Result code in AX:
- Always 0 for software SPI
- 1, if illegal arguments were passed
- Else status bitmap:
  - BIT5 = 1: A collision has occurred between multiple masters
  - BIT6 = 1: An overrun was detected

## SC1x Return Value

AX = 0

## SC1x3/SC23 Comments

Overrun detection only works with hardware SPI and if the SLVSEL pin is used as chipselect.

## Related Topics

**Init SPI (master interface)**
**Write SPI Block (master interface)**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0x97: SPI Master: Read and Write Block

Reads n bytes from the SPI Input pin and write n bytes out to the SPI Output pin simultaneously.

## Parameters

*AH*
Must be 0x97

*DS:DI*
Pointer to buffer for storing read data

*ES:SI*
Pointer to buffer which stores write data

*CX*
Number of bytes to read and write

## Return Value

Result code in AX:
- Always 0 for software SPI
- 1, if illegal arguments were passed
- Else status bitmap:
  - BIT5 = 1: A collision has occurred between multiple masters
  - BIT6 = 1: An overrun was detected

## SC1x3/SC23 Comments

Overrun detection only works with hardware SPI and if the SLVSEL pin is used as chipselect.

## Related Topics

**Init SPI (master interface)**
**Write SPI Block (master interface)**
**Read SPI Block (master interface)**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0xA0: SPI Slave: Initialize the slave interface

Initialize the SPI slave interface.

To use the SPI slave interface the user has to install an interrupt service routine for the SPI interrupt. An interrupt occurs when the SLVSEL pin transitions from active to inactive.

## Parameters

*AH*
Must be 0xA0

*BX*
Mode
BIT0-1 = SPI mode (0-3)
BIT2 = 1 use Open-Drain-Mode, 0 use Output-Mode
BIT3 = must be 0

BIT4 = 1 invert SLVSEL chipselect pin (active high), 0 do not invert SLVSEL pin (active low)
BIT5-7 = must be 0
BIT8 = 0 Normal I/O pinout, 1 Alternate I/O pinout (swap SDI and SDO pin)
BIT9-15 = must be 0

## Return Value

none

## Comments

When operating as a slave, the SPI clock signal (SCLK) must be slower than 1/8th of the CPU clock (1/16th is recommended).

A Wired-OR mode is provided which allows multiple masters to collide on the bus without risk of damage. In this mode, an external pull-up resistor is required on the Master Out Slave In (MOSI) and Master In Slave Out (MISO) pins. Generally, pull-ups are recommended on all of the external SPI signals to ensure they are held in a valid state, even when the SPI interface is disabled.

Alternate I/O pinout means that TX and RX lines are swapped.

## Related Topics

**Read SPI (slave interface)**
**Write SPI (slave interface)**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0xA5: SPI Slave: Write

Write a byte to the SPI slave interface.

## Parameters

*AH*
Must be 0xA5

*BL*
Data to write

## Return Value

none

## Related Topics

**Init SPI (slave interface)**

Read SPI (slave interface)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0xA6: SPI Slave: Read

Read a byte from the SPI slave interface.

## Parameters

*AH*

Must be 0xA6

## Return Value

AL contains read data

## Related Topics

Init SPI (slave interface)
Write SPI (slave interface)

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAA service 0xA7: SPI Slave: Read and Write

Read and write a byte from/to the SPI slave interface.

## Parameters

*AH*

Must be 0xA7

*BL*

Data to write

## Return Value

AL contains read data

## Related Topics

**Init SPI (slave interface)**
**Read SPI (slave interface)**
**Write SPI (slave interface)**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAA service 0xA8: SPI Master: Wait on / Release semaphore

Wait on semaphore / Release semaphore
If multiple tasks use the SPI hardware master interface to access different slaves through different slave chip select PIO pins, this call should be used to synchronise the SPI activity on the bus. The external MMC/SD driver examples uses this call since his version V1.02.

## Parameters

*AH*
Must be 0xA8

*BX*
0: Release the semaphore
1: Wait on semaphore
2: Wait on semaphore and set SPI mode and divider

*CX*
SPI mode (see **SPI init function**)

*SI*
SPI divider (see **SPI init function**)

## Return Value

AX contains **error** code

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

**Top of list**
**Index page**




# @CHIP-RTOS - Ethernet Packet Driver Interface



## Ethernet: Packet Driver Interface

API definition for access the ethernet device Packet driver interface definition for direct access to the IPC@CHIP® Ethernet device.

Packet drivers provide a simple, common programming interface that allows multiple applications to share a network interface at the data link level. The packet driver demultiplex incoming packets among the applications by using the network media's standard packet type. The packet driver provides calls to initiate access to a specific packet type, to end access to it, to send a packet,and to get information about the interface. The implemented services are only a small subset of the **common packet driver interface**.

The IPC@CHIP® RTOS 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.

- **0x01: _DRIVER_INFO, Get Driver Information**
- **0x02: _ACCESS_TYPE, Install Access Handler**
- **0x03: _RELEASE_TYPE, Unload an Access Handler**
- **0x04: _SEND_PKT, Send an Ethernet packet**
- **0x06: _GET_ADDRESS, Get the IPC@CHIP® Ethernet address**
- **0x14: _SET_RCV_MODE, Set receive mode**
- **0x15: _GET_RCV_MODE, Get current receive mode**
- **0x16: _SET_MULTICAST, Set Ethernet multicast address**
- **0x27: _DEL_MULTICAST, Remove Ethernet multicast address**
- **0x28: _INSTALL_WILDCARD, Install 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            // Insufficient space
0x0A    TYPE_INUSE          // Type already in use
0x0B    BAD_COMMAND         // Command not supported
0x0C    CANT_SEND           // Packet couldn't be sent
                            // (hardware error?).
```

**Important:**

In all @CHIP-RTOS variants except the TINY version, ARP and IP handlers cannot be set, as these are handled by the IPC@CHIP®RTOS. In the variant TINY the IPC@CHIP®RTOS does not handle any ethernet packets, therefore the user program can handle ARP and IP packets. One important difference

to other standard packet drivers 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 function0x16 to check if the packet driver interface is available in the IPC@CHIP®@CHIP-RTOS.

The maximum number of installed packet handlers is five.
Source code examples for the usage of the packet interface can be found at www.beck-ipc.com.

---

## Interrupt 0xAE service 0x01: DRIVER_INFO, Get Driver Information

Provided 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, "SC1xPKT"

### Related Topics

ACCESS_TYPE Install Access Handler

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAE service 0x02: ACCESS_TYPE, Install Access Handler

### Parameters

*AH*
0x02

*CX*
type length, must be 2!

*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

```
5    BAD_TYPE       - Type not allowed or CX was not set to 2.
9    NO_SPACE       - Limit of 5 accesses reached
10   TYPE_INUSE     - The type is already accessed
```

## Comments

We support the following Ethernet frame format (802.3):

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 IPC@CHIP®'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 per accepted packet:

- 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. If this pointer is NULL then no second call will be made by the driver and the packet will be discarded.
- Second call:
  Input parameters to your handler: AX = 1, CX = received packet length and data is now ready in your buffer
  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.

**SC1x3/SC23 Comment**

For SC1x3/SC23 systems, the Ethernet packet is immediately available on the first callback, where AX is zero. A pointer to this packet is passed to the callback in registers [DX:SI]. This referenced packet buffer is volatile and is only valid up to the return from your callback, so you must make a copy of the packet data inside your callback if you want the data. Return with register ES set to zero if you do not want a second call performed following a copy operation to a buffer at [ES:DI]. When you set ES to zero, no copy operation is performed by the @Chip-RTOS.

Note that the two callback operation may still be used in same manner as for the SC1x systems, if desired. It is not necessary that you use this additional "packet at [DX:SI]" feature which is available only on the SC1x3/SC23 targets.

**Important:**

Because of our Little Endian processor you must exchange the bytes for the packet types e.g. use 0x0008 for the IP type instead of 0x0800.

**Simple example of an receiver callback function:**

```
void interrupt receiver (unsigned bp, unsigned di, unsigned si,
                         unsigned ds, unsigned es, unsigned dx,
                         unsigned cx, unsigned bx, unsigned ax,
                         unsigned ip, unsigned cs, unsigned flags )
{
  if (ax == 0) // ax==0 -> Packet driver asks for a buffer
  {            // We provide the buffer at ES:DI
    es = FP_SEG(&receivepacket);
    di = FP_OFF(&receivepacket);
  }

  if (ax == 1) //Packet driver has filled our provided buffer
  {
    //Call our external c-function packet_receive to process the received data
    packet_received(cx);
  }
}
```

## Related Topics

**RELEASE_TYPE** Unload an Access Handler
**INSTALL_WILDCARD** Install wildcard access handler

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V1.05 | V1.00 |

**Top of list**
**Index page**

---

## 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

## Related Topics

ACCESS_TYPE Install Access Handler

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## 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 (min. length 60)

*DS:SI*
Pointer to packet buffer

## Return Value

Carry flag:0, AX:0, DX:0: success
Carry flag:1, DH contains error code

```
      12    CANT_SEND    - Send failed
```

## Comments

The data packet buffer should have the Ethernet packet form defined by IEEE 802.3:

```
typedef struct MY_PACKET_
{
    unsigned char dest[6];     // Destination MAC address
    unsigned char src[6];      // Source MAC address
    unsigned int type;         // Packet type (big endian)
    unsigned char data[DATA_SIZE];
} MY_PACKET ;
```

where `DATA_SIZE` can range up to 1500 bytes.

The `src` field can be set using the GET_ADDRESS API.

## Related Topics

GET_ADDRESS Get the Ethernet address

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAE service 0x06: GET_ADDRESS, Get the IPC@CHIP® Ethernet address

Get the IPC@CHIP® 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.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xAE service 0x14: SET_RCV_MODE, Set receive mode

Set receive mode of IPC@CHIP® Ethernet device.

## Parameters

*AH*
0x14

*CX*
Receive mode
3: Receiving packets with own Ethernet address or broadcast address
6: Promiscuous mode (receive all)

## Return Value

Carry flag:0, AX:0, DX:0: success
Carry flag:1, AX:-1;DX: 0x0B00 Bad parameter in CX

## Comments

Default receive mode is 3.
Receive mode 6 should only be used in the IPC@CHIP® Tiny version (@Chip-RTOS without the TCP/IP stack).
Calling **SET_MULTICAST** overrides the current mode with mode 5.
API call **Get receive mode** returns the current mode.

## Related Topics

**GET_RCV_MODE** Get receive mode

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAE service 0x15: GET_RCV_MODE, Get current receive mode

Get current receive mode of the Ethernet device.

## Parameters

*AH*
0x15

## Return Value

Carry flag:0, AX:3 accept any incoming packet with own address or broadcast address
AX:5 accept any incoming packet with own address, broadcast address or installed multicast addresses
AX:6 accept all incoming packets

## Related Topics

**SET_RCV_MODE** Set receive mode
**SET_MULTICAST** Set Ethernet multicast address

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAE service 0x16: SET_MULTICAST, Set Ethernet multicast address

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

```
      9     NO_SPACE     - CX was not set to 6.
```

## Comments

The address will be installed inside the Ethernet device of the IPC@CHIP®. A maximum of 64 addresses can be installed.

## Related Topics

**DEL_MULTICAST** Remove Ethernet multicast address

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAE service 0x27: DEL_MULTICAST, Remove Ethernet multicast address

Remove the 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

## Related Topics

**SET_MULTICAST** Set Ethernet multicast address

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xAE service 0x28: INSTALL_WILDCARD, Install wildcard access handler

Installs a wildcard packet type access handler.

## Parameters

*AH*
0x28

*CX*
type length, must be 2!!

*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
Carry flag :1 Failure, DH contains error code 5 due to CX was not zero

```
5    BAD_TYPE      - Input parameter CX was not set to 2.
```

## Comments

Any incoming Ethernet packet will be accepted for which one or more bits of the packet type matches to the bits of the installed wildcard packet type. If a packet type of 0xFFFF is installed, any incoming packet will be accepted.

Only one wildcard can be installed at a time. Installing a new wildcard type overwrites the previous.

Delete a wild card handle by installing a wildcard with the packet type 0x0000. "Normal" handlers installed with **ACCESS_TYPE** API are not overwritten by installing a wildcard

## Related Topics

**ACCESS_TYPE** Install Access Handler

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

End of document

---




# @CHIP-RTOS - USB API

IPC@CHIP® Documentation **Index**

## Introduction

This documentation describes the low-level software-interrupt-based API of the IPC@CHIP® RTOS dual-mode USB-driver. The SC1x3/SC23 disposes of a USB-controller that is compliant with the **USB specification version 1.1**.
Though the USB-API hides much of the complexity of the USB, one should fly over the specification to get a grasp of how USB works.
We also recommend to take notice of the **Requirements and Recommendations for USB Products with Embedded Hosts and/or Multiple Receptacles**.

The USB-driver can be used in Device mode to implement a USB device (resp. function) which will then be connected to e.g. a PC-Host; or it can be used in Host mode. In this case USB devices can be connected to the IPC@CHIP® handled by device-specific drivers that use the USB-API.
The RTOS USB-driver can operate in only one mode at a time. To implement a USB Function (Device) using this API, an additional circuit is needed, which manages attach- and speed-detection. A diagram is included in the hardware manual of the SC1x3/SC23 showing how to prepare your hardware to operate in both modes. Any PIO can be used to control this circuit. The API switches the PIO as needed.

Please note that there's no Hub support in Host mode and that there are some restrictions regarding data structures handled by the RTOS USB-driver necessary to keep the driver small and fast. These mainly concern Configurations, Interfaces and Alternate Settings. These restrictions should not constrain you much when implementing a USB-device and most devices connected to an IPC@CHIP® operating in Host mode will fulfil the conditions too.

- There can only be one Configuration. (It is uncommon to use more than one Configuration though the specification allows it.)
- There can be at most 10 Interfaces.
- Each interface can have maximal 12 alternate settings.
- Alternate settings in Device mode can only be used to alter the maximum packet size and the polling interval of its assigned endpoints. It is impossible to change e.g. the number of assigned endpoints. This would also be very uncommon. In Host mode a device may also have a different number of endpoints and different transfer types in Alternate Settings.

Use of the USB API C library is recommended, rather than dealing directly with these low level assembly lanaguage interfaces specified here.

## Topics

USB API **Data Structures**
USB API **Error Codes**

## Common Functions - Host and Device mode


- 0x00: Install dynamic link, then call function
- 0x01: Send data
- 0x02: Hand a receive buffer back to the driver
- 0x03: Wake up suspended nodes
- 0x04: Enable/disable notification of SOF events


## Device mode only Functions


- 0x10: Initialise driver for Device mode
- 0x11: Deinitialise Device mode driver
- 0x12: Register an interface
- 0x13: Register an Alternate Setting
- 0x14: Get numbers of endpoints associated with an interface
- 0x15: Configure an endpoint
- 0x16: Add a class- or verndor-specific descriptor
- 0x17: Start the Device mode driver
- 0x18: Stop the Device mode driver
- 0x19: Set STALL condition for an endpoint
- 0x1A: Enable/disable an endpoint


## Host mode only Functions


- 0x20: Initialise driver for Host mode
- 0x21: Deinitialise Host mode driver
- 0x22: Issue token
- 0x23: Clear an endpoint's queues
- 0x24: Reserve endpoint 0
- 0x25: Release endpoint 0
- 0x26: Get Device information
- 0x27: Get Configuration information
- 0x28: Get Interface information
- 0x29: Get Endpoint information
- 0x2A: Get string
- 0x2B: Get status
- 0x2C: Set/clear feature
- 0x2D: Set interface


---

## Interrupt 0xBF service 0x00: Install dynamic link, then call function

This function installs a far JMP opcode in the vicinity of the return instruction pointer to enter the specified USB API function directly from the user code. This function is provided for efficiency in real-time applications.

## Parameters

*AH*

USB_SERVICE_DYNAMIC_LINK (0x00)

*AL*

Number of USB API function that is to be linked

## Return Value

Instruction Pointer is moved back 10 bytes so that newly installed far JMP will execute immediately.

(Refer to respective linked function's documention for further return value information.)

Note: All dynamically linked functions are free to use all registers except for DS, SI, DI, and BP registers which are preserved. This matches Paradigm C, Borland C++ 5.02 and Microsoft Visual C 1.52 calling conventions.

## Comments

A JMP FAR instruction will be written 10 bytes before the user's return instruction pointer. This far jump will transfer control directly into the RTOS from the user code when executed. This new far JMP will execute immediately upon return from this software interrupt due to the instruction pointer adjustment performed within the call.

Interrupts are masked and restored to their state on entry.

Since the code at calling site will be patched by this RTOS call, it is important that it has exactly the following form (the link index carried in AL register will vary)

```
JMP SHORT over_patch  ; Occupies 2 bytes of instruction code here
NOP    ; Never executed (1 byte)
NOP    ; Never executed (1 byte)
NOP    ; never executed (1 byte)
over_patch:  ; Linker overwrites above 5 bytes with JMP FAR instruction
MOV AX,00001h  ; AH = 0x00, AL = 0x01 = usbSend()
INT 0BFh
```

## Developer Notes

This API service is intended to be used by the RTOS C-Library exclusively.
It should not be used within user applications.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x01: Send data

Enqueues a send request to an endpoint's TX queue. In Device mode the data will be transmitted when the Host issues an IN token. In Host mode the data will be transmitted when a SETUP or OUT token is issued using **USB_SERVICE_HOST_ISSUE_TOKEN**. When the request has been completed the **callback** -function will be called with a status of USB_EVENT_SENT.

## Parameters

*AH*

USB_SERVICE_SEND (0x01)

*AL*
Host mode: Address of the device that is to receive the data
Device mode: Must always be 0x00

*CL*
Number of the enpoint via which the data is to be sent

*ES:BX*
Data to be sent

*DX*
Length of data to be sent

*CH*
Host mode: Must always be FALSE
Device mode: Set this parameter to TRUE if the host demanded more data than is to be sent. This does only apply to control endpoints.

## Return Value

**Error code** in AX

## Comments

The buffer that is used for sending must be allocated by the application and be available while the USB-driver processes the send-request. After it has completed the callback-function will be called with an `USB_EVENT_SENT` event (or maybe an error event). After that the buffer can be released.

## Beck C-Library Usage

**usbSend()**

## Related Topics

**USB_SERVICE_DEVICE_CONFIG_EP** Configure an endpoint
**USB_SERVICE_HOST_ISSUE_TOKEN** Issue token
**USB_SERVICE_HOST_CLEAR_QUEUE** Clear an endpoint's queues
**USB_SERVICE_HOST_RESERVE_EP0** Reserve endpoint 0
**USB_SERVICE_HOST_RELEASE_EP0** Release endpoint 0
Type **UsbCallback**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x02: Hand a receive buffer back to the driver

Frees a receive buffer. The RTOS USB-driver manages a bunch of receive buffers internally. Each time data is received on any endpoint, it is put into one of these buffers and the buffer is handed to a user driver along with a callback event.
To enable the reuse of the buffer, this function has to be called as soon as possible.

## Parameters

*AH*
USB_SERVICE_FREE_RECEIVE_BUFFER (0x02)

*ES:BX*
Pointer to the receive buffer that is to be freed

## Return Value

None

## Beck C-Library Usage

**usbFreeReceiveBuffer()**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x03: Wake up suspended nodes

Sends a resume signal to the bus to wakeup suspended nodes

## Parameters

*AH*
USB_SERVICE_WAKEUP (0x03)

## Return Value

**Error code** in AX

## Comments

As well hosts as devices may perform resume signaling. But the Remote Wakeup Feature of devices may be disabled. In this case a call to this function will fail in Device mode.

## Beck C-Library Usage

**usbWakeup()**

## Related Topics

**USB_SERVICE_HOST_SET_CLEAR_FEATURE** Set/clear feature

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x04: Enable/disable notification of SOF events

Use this function to enable or disable the forwarding of SOF events to your application. By default the event **USB_EVENT_SOF** is not notified.

## Parameters

*AH*

USB_SERVICE_SOF_NOTIFY (0x04)

*AL*

Host mode: ID of the user driver
Device mode: Must always be 0x00

*BL*

Indicates whether to enable (TRUE) or disable (FALSE) the notification of SOF events

## Return Value

**Error code** in AX

## Beck C-Library Usage

**usbSofNotify()**

## Related Topics

Type **UsbCallback**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x10: Initialise driver for Device mode

Initialises the USB driver in Device mode and registers a **callback**-function that will be called to notify the user driver of events like completed transactions.
Also sets sundry strings that describe the device.
Only one user program can use the USB driver in Device mode.
Several information on the device is given to this function via the structure **UsbInit**. For detailed information on the structure's members refer to its documentation.

## Parameters

*AH*
USB_SERVICE_DEVICE_INIT (0x10)

*ES:BX*
Pointer to the structure variable

## Return Value

**Error code** in AX

## Beck C-Library Usage

**usbDeviceInit()**

## Related Topics

**USB_SERVICE_DEVICE_DEINIT** Deinitialise Device mode driver
Type **UsbInit**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x11: Deinitialise Device mode driver

This function deinitialises the USB-driver in Device mode.

## Parameters

*AH*
USB_SERVICE_DEVICE_DEINIT (0x11)

## Return Value

None

## Comments

If the USB-driver has already been started, stop it first.

## Beck C-Library Usage

usbDeviceDeinit()

## Related Topics

USB_SERVICE_DEVICE_INIT Initialise driver for Device mode
USB_SERVICE_DEVICE_STOP Stop the Device mode driver

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x12: Register an interface

This function registers a new interface.

## Parameters

*AH*
USB_SERVICE_DEVICE_REGISTER_INTERFACE (0x12)

*AL*
Number of endpoints to associate with the interface

*CL*
Class code of this interface

*DL*
Subclass code of this interface

*CH*
Protocol code of this interface

*ES:BX*
Human readable string describing this interface; must be static for the RTOS USB-driver will not copy it. The string can have a maximum length of USB_SPEC_MAX_STRING_LENGTH characters. If no string is needed, set this to NULL.

## Return Value

ID of the newly registered interface or Error code in AX

## Comments

For further information on Class, Subclass and Protocol code see the description of

USB_SERVICE_DEVICE_INIT .
Use the function USB_SERVICE_DEVICE_GET_EPS to get the numbers of the endpoints that are assigned to the interface.

## Beck C-Library Usage

usbDeviceRegisterInterface()

## Related Topics

USB_SERVICE_DEVICE_INIT Initialise driver for Device mode
USB_SERVICE_DEVICE_REGISTER_ALT_SETTING Register an Alternate Setting
USB_SERVICE_DEVICE_GET_EPS Get numbers of endpoints associated with an interface
USB_SERVICE_DEVICE_ADD_SPECIFIC_DESC Add a class- or verndor-specific descriptor

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x13: Register an Alternate Setting

This function registers a new alterate setting to a given interface.

## Parameters

*AH*
USB_SERVICE_DEVICE_REGISTER_ALT_SETTING (0x13)

*AL*
ID of the interface to that the alternate settings is to be registered

## Return Value

ID of the new alternate setting or Error code in AX

## Comments

Alternate settings can only be used to change the maximum packet size of associated endpoints and the polling interval of associated interrupt or isochronous endpoints. Alternate settings cannot be used to alter the number or transfer type of endpoints.

## Beck C-Library Usage

usbDeviceRegisterAltSetting()

## Related Topics

USB_SERVICE_DEVICE_REGISTER_INTERFACE Register an interface
USB_SERVICE_DEVICE_CONFIG_EP Configure an endpoint

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x14: Get numbers of endpoints associated with an interface

This function gets the numbers of the endpoints that have been associated with an interface. The numbers are returned in a bitmap field. Each bit that is set to "1" marks an assigned endpoint. If e.g. Bit 1 is set, endpoint 1 is assigned to the interface.
By default both the IN and OUT endpoint at the resp. endpoint number are reserved to the interface. You decide which direction actually to assign to the interface when configuring the endpoints. You may even use both directions if both have the same transfer type.

## Parameters

*AH*
USB_SERVICE_DEVICE_GET_EPS (0x14)

*AL*
ID of the interface whose endpoints are to be gotten

*ES:BX*
Result parameter: Bitmap indicating which endpoints belong to the given interface.

## Return Value

Error code in AX

## Beck C-Library Usage

usbDeviceGetEps()

## Related Topics

USB_SERVICE_DEVICE_REGISTER_INTERFACE Register an interface
USB_SERVICE_DEVICE_CONFIG_EP Configure an endpoint

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list

## Interrupt 0xBF service 0x15: Configure an endpoint

This function can be used to configure an endpoint. Transfer type and the transfer direction can be configured once for each endpoint. The maximum packet size and the polling interval can be configured for each alternate setting of the interface that the endpoint is associated to.

### Parameters

*AH*

USB_SERVICE_DEVICE_CONFIG_EP (0x15)

*AL*

ID of the interface the endpoint is associated to

*BL*

ID of the alternate setting to the given interface

*CL*

Number of the endpoint that is to be configured

*BH*

Transfer type; possible values are:

- `USB_SPEC_TRANSFER_TYPE_CONTROL`
- `USB_SPEC_TRANSFER_TYPE_ISOCHRONOUS`
- `USB_SPEC_TRANSFER_TYPE_BULK`
- `USB_SPEC_TRANSFER_TYPE_INTERRUPT`

The symbolic names are defined in USB_SPEC.H.

*CH*

TRUE, if this is an IN-endpoint, which means that data is transfered from the device to the host. This parameter is ignored for control-endpoints.

*DX*

Maximum size of packets sent/received through this endpoint; Please note that the packet size may also be limited by the USB specification and by the capabilities of the controller.

*ES*

Polling interval for interrupt and isochronous endpoints; given in 2ms steps.

### Return Value

Maximum packet size set (respecting the USB specification and the capabilities of the controller) or **Error code** in AX

### Beck C-Library Usage

**usbDeviceConfigEp()**

### Related Topics

**USB_SERVICE_DEVICE_REGISTER_INTERFACE** Register an interface
**USB_SERVICE_DEVICE_REGISTER_ALT_SETTING** Register an Alternate Setting
**USB_SERVICE_DEVICE_GET_EPS** Get numbers of endpoints associated with an interface

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x16: Add a class- or verndor-specific descriptor

This function can be used to add class- oder vendor-specific descriptors to the Configuration, to an interface or an endpoint. A buffer has to be allocated by the application into which the descriptor is written. Several descriptors can be written in this single buffer, but only one buffer can be added to each entity. The buffer must be available during the whole program execution because the RTOS USB-driver will not copy it. The specific descriptors will be concatenated to the resp. standard descriptors.

## Parameters

*AH*

USB_SERVICE_DEVICE_ADD_SPECIFIC_DESC (0x16)

*AL*

Type of entity to which the specific descriptor is to be added. These are possible values:

- `USB_SPEC_DESCRIPTOR_TYPE_CONFIG`
- `USB_SPEC_DESCRIPTOR_TYPE_INTERFACE`
- `USB_SPEC_DESCRIPTOR_TYPE_EP`

The symbolic names are defined in USB_SPEC.H.

*CL*

ID/number of entity to which the specific descriptor is to be added. If the descriptor type is `USB_SPEC_DESCRIPTOR_TYPE_EP` this is the endpoint number. If descriptor type is `USB_SPEC_DESCRIPTOR_TYPE_INTERFACE` this is the index of the interface. If descriptor type is `USB_SPEC_DESCRIPTOR_TYPE_CONFIG` this parameter should be 0.

*ES:BX*

Pointer to the buffer containing the descriptor(s)

*DL*

Length of the descriptor(s) in bytes

## Return Value

Error code in AX

## Comments

If a specific descriptor is added to an interface it is added to all alternate settings.

## Beck C-Library Usage

usbDeviceAddSpecificDesc()

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x17: Start the Device mode driver

This function will start the Device mode USB-driver and enable the attach- and speed-detection circuit via the configured PIO pin.

### Parameters

*AH*
USB_SERVICE_DEVICE_START (0x17)

### Return Value

None

### Beck C-Library Usage

usbDeviceStart()

### Related Topics

USB_SERVICE_DEVICE_STOP Stop the Device mode driver

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x18: Stop the Device mode driver

This function will stop the Device mode USB-driver and disable the attach- and speed-detection circuit via the configured PIO pin.

### Parameters

*AH*
USB_SERVICE_DEVICE_STOP (0x18)

### Return Value

None

## Beck C-Library Usage

usbDeviceStop()

## Related Topics

USB_SERVICE_DEVICE_START Start the Device mode driver
USB_SERVICE_DEVICE_DEINIT Deinitialise Device mode driver

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x19: Set STALL condition for an endpoint

This function will set the STALL condition for an endpoint. The USB controller will reject all arriving packets and send requests on this endpoint and return a STALL handshake, which tells the Host that the endpoint has a permanent error. A distinction between the IN and OUT direction is not possible. Both directions will be stalled.
Host intervention is needed to reset the STALL condition.

## Parameters

*AH*
USB_SERVICE_DEVICE_STALL_EP (0x19)

*AL*
Number of endpoint that is to be stalled

## Return Value

Error code in AX

## Beck C-Library Usage

usbDeviceStallEp()

## Related Topics

USB_SERVICE_HOST_SET_CLEAR_FEATURE Set/clear feature

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Interrupt 0xBF service 0x1A: Enable/disable an endpoint

This function enables or disables communication in the OUT direction (Host to Device) of an endpoint. When the Host tries to send data on a disabled endpoint a NAK handshake will be returned (except for ISOCHRONOUS endpoints). A NAK handshake tells the Host that the Device is temporary unable to receive data.

By default all configured endpoints are enabled after the Device has been started.

Note that this function will not stop reception immediately. At least two more packets can be received, because respective requests have already been handed to the controller hardware. Some more data may already have been received, but still be pending in a software queue inside the RTOS.

## Parameters

*AH*

USB_SERVICE_DEVICE_ENABLE_EP (0x1A)

*AL*

Number of endpoint that is to be enabled/disabled

*BH*

Indicates whether the endpoint is to be enabled (TRUE) or disabled (FALSE)

## Return Value

Error code in AX

## Comments

If the device is permanent unable to receive data, the endopint should be stalled.

To achieve sending of a NAK handshake in the IN direction (Device to Host) of an endpoint, simply stop sending on this endpoint. If the Host then tries to receive data on that endpoint, a NAK handshake is returned automatically.

## Beck C-Library Usage

usbDeviceEnableEp()

## Related Topics

USB_SERVICE_DEVICE_STALL_EP Set STALL-condition for an endpoint

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.20 | V1.20 |

## Interrupt 0xBF service 0x20: Initialise driver for Host mode

This function registers a user driver and initialises the RTOS USB-driver for Host mode if no other user driver has already done this. There can be several user drivers loaded in Host mode. Each driver registers a **callback**-function which is called when an event (e.g. completed transaction) appears on a device that the driver is assigned to. User drivers are assigned to Interfaces which allows a device with two different functions and thus two different Interfaces to be handled by two different drivers. The criteria for the assignment of user drivers to Interfaces are the codes handed to this function.

## Parameters

*AH*
USB_SERVICE_HOST_INIT (0x20)

*ES:BX*
Callback function

*AL*
Class code

*CL*
Subclass code

*CH*
Protocol code
If the user driver can handle devices of one of the Device Classes, these arguements identify the class, subclass and protocol. If the user driver handles vendor-specific protocols, set these arguments to 0.

*DX*
Vendor ID

*SI*
Product ID

*DI*
Product version in BCD code
If the user driver handles devices using a vendor-specific protocol, these arguements identify the Vendor and Product ID of the resp. device as well as the maximum product version that the driver can handle. If the user driver can handle devices of one of the Device Classes, set these arguments to 0.

## Return Value

Assigned driver ID or **error code** in AX; The driver ID will be needed for other API functions.

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

**usbHostInit()**

## Related Topics

USB_SERVICE_HOST_DEINIT Deinitialise Host mode driver

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x21: Deinitialise Host mode driver

This function unregisters a user driver and deinitialises the USB driver if there are no more user drivers registered.

## Parameters

*AH*
USB_SERVICE_HOST_DEINIT (0x21)

*AL*
ID of the user driver that is to be unregistered

## Return Value

None

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

usbHostDeinit()

## Related Topics

USB_SERVICE_HOST_INIT Initialise driver for Host mode

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x22: Issue token

Issues an IN, OUT or SETUP token on a given endpoint. This is needed in Host mode to initiate a data transfer. To receive a packet from a device, an IN token has to be issued to tell the device to start sending. To send a packet to a device an OUT or SETUP token has to be issued after calling **USB_SERVICE_SEND** to tell the RTOS USB-driver to start transmission.

## Parameters

*AH*

USB_SERVICE_HOST_ISSUE_TOKEN (0x22)

*AL*

Address of the device that is to receive the token

*BL*

Number of the endpoint that is to receive the token

*BH*

Token that is to be sent; must be one of the following:

- `USB_SPEC_PID_IN`
- `USB_SPEC_PID_OUT`
- `USB_SPEC_PID_SETUP`

The symbolic names are defined in USB_SPEC.H.

## Return Value

**Error code** in AX

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

**usbHostIssueToken()**

## Related Topics

**USB_SERVICE_SEND** Send data

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x23: Clear an endpoint's queues

This function clears the send and token queue of a given endpoint. It can be used to reset an endpoint when an error occured.
Buffers associated with send requests cleared by this function will later be handed back to the user driver via an event with the status **USB_EVENT_RELEASE**.

Note that queue entries already passed to the USB controller cannot be removed.

## Parameters

*AH*

USB_SERVICE_HOST_CLEAR_QUEUE (0x23)

*AL*

Address of the resp. device

*BL*

Number of the endpoint whose queues are to be cleared

*BH*

States that the IN direction of the endpoint is to be cleared; set to FALSE for OUT or CONTROL endpints

## Return Value

**Error code** in AX

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

**usbHostClearQueue()**

## Related Topics

**USB_SERVICE_SEND** Send data
**USB_SERVICE_HOST_ISSUE_TOKEN** Issue token

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x24: Reserve endpoint 0

Reserves endpoint 0 of a device for a user driver. Since it might happen that several user drivers share one device, access to endpoint 0 of the device has to be synchronised. A user driver must not use (i.e. send or issue tokens) endpoint 0 before reserving it.
The endpoint has to be released again using the interrupt service **USB_SERVICE_HOST_RELEASE_EP0**.

## Parameters

*AH*

USB_SERVICE_HOST_RESERVE_EP0 (0x24)

*AL*

ID of the user driver

*BL*

Address of the resp. device

## Return Value

**Error code** in AX

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

**usbHostReserveEp0()**

## Related Topics

Interrupt service **USB_SERVICE_HOST_RELEASE_EP0**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x25: Release endpoint 0

Releases endpoint 0 of a device after it was reserved using the interrupt service **USB_SERVICE_HOST_RESERVE_EP0**.

## Parameters

*AH*

USB_SERVICE_HOST_RELEASE_EP0 (0x25)

*AL*

ID of the user driver that has reserved the endpoint

*BL*

Address of the resp. device

## Return Value

**Error code** in AX

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

usbHostReleaseEp0()

## Related Topics

Interrupt service USB_SERVICE_HOST_RESERVE_EP0

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x26: Get Device information

Returns information from the Device Descriptor of a given device

## Parameters

*AH*
USB_SERVICE_HOST_GET_DEVICE_INFO (0x26)

*AL*
Address of the resp. device

*ES:BX*
Result parameter: Pointer to a structure of the type UsbDevice which is to be filled with the information

*CL*
Size of the handed structure in bytes

## Return Value

Error code in AX

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

usbHostGetDeviceInfo()

## Related Topics

**USB_SERVICE_HOST_GET_CONFIG_INFO** Get Configuration information
**USB_SERVICE_HOST_GET_INTERFACE_INFO** Get Interface information
**USB_SERVICE_HOST_GET_EP_INFO** Get Endpoint information

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x27: Get Configuration information

Returns information from the Configuration Descriptor of a given device

## Parameters

*AH*
USB_SERVICE_HOST_GET_CONFIG_INFO (0x27)

*AL*
Address of the resp. device

*ES:BX*
Result parameter: Pointer to a structure of the type **UsbConfig** which is to be filled with the information

*CL*
Size of the handed structure in bytes

## Return Value

**Error code** in AX

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

**usbHostGetConfigInfo()**

## Related Topics

**USB_SERVICE_HOST_GET_DEVICE_INFO** Get Device information
**USB_SERVICE_HOST_GET_INTERFACE_INFO** Get Interface information
**USB_SERVICE_HOST_GET_EP_INFO** Get Endpoint information

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x28: Get Interface information

Returns information from the Interface Descriptor of a given interface

## Parameters

*AH*
USB_SERVICE_HOST_GET_INTERFACE_INFO (0x28)

*AL*
Address of the resp. device

*CL*
ID of the resp. Interface

*CH*
ID of the resp. Alternate Setting

*ES:BX*
Result parameter: Pointer to a structure of the type **UsbInterface** which is to be filled with the information

*DL*
Size of the handed structure in bytes

## Return Value

**Error code** in AX

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

**usbHostGetInterfaceInfo()**

## Related Topics

**USB_SERVICE_HOST_GET_DEVICE_INFO** Get Device information
**USB_SERVICE_HOST_GET_CONFIG_INFO** Get Configuration information
**USB_SERVICE_HOST_GET_EP_INFO** Get Endpoint information

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x29: Get Endpoint information

Returns information from the Endpoint Descriptor of a given endpoint

## Parameters

*AH*
USB_SERVICE_HOST_GET_EP_INFO (0x29)

*AL*
Address of the resp. device

*CL*
ID of the resp. alternate setting

*CH*
Number of the resp. endpoint

*DL*
Indicates that information on an IN endpoint is desired; always set to FALSE for control-endpoints

*ES:BX*
Result parameter: Pointer to a structure of the type **UsbEp** which is to be filled with the information

*DL*
Size of the handed structure in bytes

## Return Value

**Error code** in AX

## Comments

This function must not be called from within the callback-function.

## Beck C-Library Usage

**usbHostGetEpInfo()**

## Related Topics

**USB_SERVICE_HOST_GET_DEVICE_INFO** Get Device information
**USB_SERVICE_HOST_GET_CONFIG_INFO** Get Configuration information
**USB_SERVICE_HOST_GET_INTERFACE_INFO** Get Interface information

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x2A: Get string

Retrieves a string from a given device

## Parameters

*AH*
USB_SERVICE_HOST_GET_STRING (0x2A)

*AL*
ID of the user driver

*CL*
Address of the resp. device

*CH*
ID of the string that is desired

*ES:BX*
Pointer to a buffer where the string is to be stored; Must be large enough to store a string consisting of `USB_SPEC_MAX_STRING_LENGTH` characters plus trailing zero-byte

## Return Value

Error code in AX

## Comments

This function reserves endpoint 0 of the resp. device for the user driver that calls it before using the endpoint and it releases it again afterwards.
Note that this function may take up to 2000 ms to complete, because it waits that long for data to arrive from the device.
This function must not be called from within the callback-function.

## Beck C-Library Usage

usbHostGetString()

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list

## Interrupt 0xBF service 0x2B: Get status

Retrieves status information from a given device using the GetStatus Standard Device Request

## Parameters

*AH*

USB_SERVICE_HOST_GET_STATUS (0x2B)

*AL*

ID of the user driver

*CL*

Address of the resp. device

*CH*

Receiver of the GetStatus request; this parameter decides whether the status of the device, an interface, an endpoint or a user- defined structure is to be retrieved; must be one of the following:

- `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_DEVICE`
- `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_INTERFACE`
- `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_ENDPOINT`
- `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_OTHER`

The symbolic names are defined in USB_SPEC.H.

*DL*

Index of the resp. structure. The value of this parameter depends on the value of the receiver parameter.
If the receiver is `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_DEVICE`, the index must always be 0.
If the receiver is `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_INTERFACE`, the index must contain the ID of the resp. interface.
If the receiver is `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_ENDPOINT`, the index parameter must contain the number of the resp. endpoint. If it is an IN endpoint, bit 7 has to be set additionally. (e.g. IN endpoint 1 => 0x81).
If the receiver is `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_OTHER`, the value of the index parameter is device-specific.

*ES:BX*

Pointer to an `unsigned short` variable where the status is to be stored.

## Return Value

Error code in AX

## Comments

This function reserves endpoint 0 of the resp. device for the user driver that calls it before using the endpoint and it releases it again afterwards.
Note that this function may take up to 2000 ms to complete, because it waits that long for data to arrive from the device.
This function must not be called from within the callback-function.

## Beck C-Library Usage

usbHostGetStatus()

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x2C: Set/clear feature

Sets or clears a feature of a given device by sending a SetFeature or ClearFeature Standard Device Request

## Parameters

*AH*

USB_SERVICE_HOST_SET_CLEAR_FEATURE (0x2C)

*AL*

ID of the user driver

*BL*

Address of the resp. device

*BH*

Receiver of the SetFeature or GetFeature request; this parameter decides whether a feature of the device, an interface, an endpoint or a user- defined structure is to be set/cleared; must be one of the following:

- `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_DEVICE`
- `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_INTERFACE`
- `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_ENDPOINT`
- `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_OTHER`

The symbolic names are defined in USB_SPEC.H.

*CL*

Index of the resp. structure. The value of this parameter depends on the value of the receiver parameter.
If the receiver is `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_DEVICE`, the index must always be 0.
If the receiver is `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_INTERFACE`, the index must contain the ID of the resp. interface.
If the receiver is `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_ENDPOINT`, the index parameter must contain the number of the resp. endpoint. If it is an IN endpoint, bit 7 has to be set additionally. (e.g. IN endpoint 1 => 0x81).
If the receiver is `USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_OTHER`, the value of the index parameter is device-specific.

*CH*

Identifies the feature that is to be set/cleared. Actually the feature `USB_SPEC_FEATURE_REMOTE_WAKEUP` is defined for devices and `USB_SPEC_FEATURE_STALL` for endpoints.
The symbolic names are defined in USB_SPEC.H.

*DL*

If this parameter is TRUE the feature will be set, otherwise it will be cleared.

## Return Value

Error code in AX

## Comments

This function reserves endpoint 0 of the resp. device for the user driver that calls it before using the endpoint and it releases it again afterwards.
Note that this function may take up to 2000 ms to complete, because it waits that long for data to arrive from the device.
This function must not be called from within the callback-function.

## Beck C-Library Usage

usbHostSetClearFeature()

## Related Topics

USB_SERVICE_WAKEUP Wake up suspended nodes
USB_SERVICE_DEVICE_STALL_EP Set STALL-condition for an endpoint

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x2D: Set interface

Sets one of the alternate settings of a given interface as the current one by sending a SetInterface Standard Device Request

## Parameters

*AH*
USB_SERVICE_HOST_SET_INTERFACE (0x2D)

*AL*
ID of the user driver

*BL*
Address of the resp. device

*BH*
ID of the resp. interface

*CL*
ID of the alternate setting that is to be activated

## Return Value

Error code in AX

## Comments

This function reserves endpoint 0 of the resp. device for the user driver that calls it before using the endpoint and it releases it again afterwards.
Note that this function may take up to 2000 ms to complete, because it waits that long for data to arrive from the device.
This function must not be called from within the callback-function.

## Beck C-Library Usage

**usbHostSetInterface()**

## Related Topics

**USB_SERVICE_HOST_GET_CONFIG_INFO** Get Configuration information
**USB_SERVICE_HOST_GET_INTERFACE_INFO** Get Interface information

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

**Top of list**
**Index page**

End of document

# @CHIP-RTOS - USB API Data Structures

IPC@CHIP® Documentation **Index**

## Introduction

In this section data structures are described that are needed with the USB API. These are defined in the header-file USB_API.H.
Notes:

- **Byte alignment is required** for all data structures used within the API.

### Component List

- **UsbEventStatus**
- **UsbCallback**
- **UsbInit**
- **UsbDevice**
- **UsbConfig**
- **UsbInterface**
- **UsbEp**

## UsbEventStatus

```
typedef enum
{
  USB_EVENT_RECEIVED = 0,
  USB_EVENT_SENT = 1,
  USB_EVENT_RESET = 2,
  USB_EVENT_SETUP = 3,
  USB_EVENT_SUSPEND = 4,
  USB_EVENT_RESUME = 5,
  USB_EVENT_SOF = 13,
  USB_EVENT_ATTACH = 14,
  USB_EVENT_DETACH = 15,
  USB_EVENT_RELEASE = 16,
  USB_EVENT_ERROR_BIT_STUFF = 6,
  USB_EVENT_ERROR_DMA = 7,
  USB_EVENT_ERROR_TURNAROUND = 8,
  USB_EVENT_ERROR_DATA_FIELD = 9,
  USB_EVENT_ERROR_CRC16 = 10,
```

```
  USB_EVENT_ERROR_CRC5 = 11,
  USB_EVENT_ERROR_PID = 12,
  USB_EVENT_NAK = 17,
  USB_EVENT_STALL = 18,
  USB_EVENT_ERROR_DATA_TOGGLE = 19,
  USB_EVENT_ERROR_SOF_LOST = 20,
  USB_EVENT_ERROR_SOF_BANDWIDTH = 21,
  USB_EVENT_ERROR_ATTACH = 22,
  USB_EVENT_ERROR_LOST_EVENT = 23,
  USB_EVENT_INVALID = -1
} UsbEventStatus;
```

## Comments

This is the type of the status parameter passed along with **callbacks**. It indicates the type of event that has occured and decides what the meaning of the other parameters is.

**USB_EVENT_RECEIVED**
Data has been received.
The `address`parameters holds the address of the device from which data was received in Host mode - 0 in Device mode.
`epNumber`indicates the endpoint on which data was received.
`buffer`holds the received data.
`length`holds the number of bytes stored in buffer.
After processing the data the function **usbFreeReceiveBuffer()**(interrupt service **USB_SERVICE_FREE_RECEIVE_BUFFER**) has to be called to free the buffer containing the data.

**USB_EVENT_SENT**
Data has been successfully sent.
`address`holds the address of the device to which data was sent in Host mode - 0 in Device mode.
`epNumber`indicates the endpont on which data was sent.
`buffer`holds the sent data.
`length`holds the number of bytes stored in buffer.

**USB_EVENT_RESET**
The device has been reset. (Device mode only)
All other callback parameters are 0.

**USB_EVENT_SETUP**
A setup token has been received. (Device mode only)
`epNumber`indicates on which endpoint a Setup token has been received. All other callback parameters are 0.
Processing of a previous control-transaction must be stopped. The received data will be notified in a seperate `USB_EVENT_RECEIVED`-event.

**USB_EVENT_SUSPEND**
The device has been suspended, i.e. there has been no activity on the bus for 3 ms. (Device mode only)
This event can be used to save power by shutting e.g. down peripherals. All other callback parameters are 0.

**USB_EVENT_RESUME**
The node has been woken up from suspension.
All other callback parameters are 0.

**USB_EVENT_SOF**
A Start-Of-Frame token has been issued, indicating the beginning of a new 1-ms-frame.
All other callback parameters are 0.
In case of this event the callback function is called directly from within the resp. interrupt service routine. Consider this when designing your software.
Use the function function **usbSofNotify()**(interrupt service **USB_SERVICE_SOF_NOTIFY**) to

enable or disable the notification of this event to your application. This event may be substituted by one of the events USB_EVENT_ERROR_SOF_LOSTor USB_EVENT_ERROR_SOF_BANDWIDTHin case of an error.

**USB_EVENT_ATTACH**
A new device has been attached. (Host mode only) This event is only notified to a user driver if it can handle the resp. device.
addressholds the address of the new device.
All other callback parameters are 0.

**USB_EVENT_DETACH**
A device has been detached. (Host mode only) This event is only notified to a user driver if it handled the resp. device.
addressholds the address of the device.
All other callback parameters are 0.

**USB_EVENT_RELEASE**
This event is used to hand a send buffer back to a user driver in Host mode.
The user driver passed this buffer to the USB driver along with a send request but then cleared the send queue for any reason (See API function **usbHostClearQueue()**, interrupt service **USB_SERVICE_HOST_CLEAR_QUEUE**).
bufferholds the pointer to the resp. user driver buffer.
All other callback parameters are 0.

The follwoing events indicate error conditions. If nothing else is stated, the callback parameter address indicates the device with which an error occured in Host mode or 0 in Device mode, epNumber indicates the resp. endpoint and length indicates whether the error occured while receiving or sending as a boolean value (TRUE = sending, FALSE = receiving).

**USB_EVENT_ERROR_BIT_STUFF**
Indicates that a bit stuff error was detected in the data stream on the bus. The resp. packet was rejected. The Host should repeat it.

**USB_EVENT_ERROR_DMA**
Fatal! There was a problem regarding DMA data transport between memory and the USB controller hardware. Data may be lost.

**USB_EVENT_ERROR_TURNAROUND**
A bus turnaround timeout occured. This means that too much time elapsed between two phases of a data transfer. This is likely to happen if the USB cable is too long.

**USB_EVENT_ERROR_DATA_FIELD**
Indicates that a data field did not consist of an integral number of bytes. The resp. packet was rejected. The Host should repeat it.

**USB_EVENT_ERROR_CRC16 or USB_EVENT_ERROR_CRC5**
Indicates a checksum error. The resp. packet was rejected. The host should repeat it.
In Host mode USB_EVENT_ERROR_CRC5indicates an internal timing error. This should not happen.

**USB_EVENT_ERROR_PID**
A PID check field didn't match. This is similar to a checksum error. The resp. packet was rejected. The Host should repeat it.

**USB_EVENT_NAK**
A transaction on an INTERRUPT endpoint has been NAKed by the device. (Host mode only)

**USB_EVENT_STALL**
The device returned a STALL handshake on the given endpoint. (Host mode only)

**USB_EVENT_ERROR_DATA_TOGGLE**
A data packet arrived with a wrong DATA PID. All other callback parameters match the description for the USB_EVENT_RECEIVEDevent. It is up to the user driver to decide whether to use or discard the package.

The three following error events do not relate to a special device or endpoint.
All callback parameters, except for the status, will be set to 0.

**USB_EVENT_ERROR_SOF_LOST**
This event indicates a SOF token (see USB_EVENT_SOF), but it also indicates that a SOF token has been lost.
In case of this event the callback function is called directly from within the resp. interrupt service routine. Consider this when designing your software.

**USB_EVENT_ERROR_SOF_BANDWIDTH**
This event indicates a SOF token (see USB_EVENT_SOF), but it also indicates that not all transactions of the last frame had been completed. If these were CONTROL or BULK transactions they will be automatically processed within the next frame. Transactions on INTERRUPT and ISOCHRONOUS endpoints are lost. (Host mode only)
In case of this event the callback function is called directly from within the resp. interrupt service routine. Consider this when designing your software.

**USB_EVENT_ERROR_ATTACH**
A newly attached device could not be configured correctly. (Host mode only)
The user driver cannot determine what kind of device had been attached.

**USB_EVENT_ERROR_LOST_EVENT**
Fatal! An overrun occured in the ring queue used to manage events and events have been lost.

## Related Topics

Type **UsbCallback**

**Top of list**
**Index page**

---

# UsbCallback

```
typedef void huge ( far *UsbCallback )( unsigned epNumber,
                                        UsbEventStatus status,
                                        void far *buffer,
                                        unsigned length,
                                        unsigned address );
```

## Comments

This is the prototype for callback-functions that a user driver registers with the USB-API. The callback-function is used to notify asynchronous events to the application, such as received data packets or completed send-requests.
The functions **usbHostInit()** (interrupt service **USB_SERVICE_HOST_INIT**) and **usbDeviceInit()** (interrupt service **USB_SERVICE_DEVICE_INIT**) are used to install a callback-function.

**epNumber**
In this parameter the endpoint number an event is assigned to is given. Of course this is not applicable for all events. In this case epNumberwill be 0.

**status**
This parameter indicates the event that has occured. Refer to **UsbEventStatus**.

**buffer**

If data has been sent or received the resp. data is given within the buffer pointed to by this parameter.
Note that the buffer must be handed back to the RTOS USB-driver using the function **usbFreeReceiveBuffer()**(interrupt service **USB_SERVICE_FREE_RECEIVE_BUFFER**) if data has been received.

**length**

Here the length of the data passed in parameter buffer is given.
For error events this parameter indicates whether the error occured while receiving or sending as a boolean value (TRUE = sending, FALSE = receiving).

**address**

In Host mode the `address`parameter holds the address of the device with that the event is associated. In Device mode or if this is not applicable, it will be 0.

## Related Topics

Type **UsbEventStatus**
API function **usbHostInit()**
Interrupt service **USB_SERVICE_HOST_INIT**
API function **usbDeviceInit()**
Interrupt service **USB_SERVICE_DEVICE_INIT**

**Top of list**
**Index page**

---

# UsbInit

```
typedef struct UsbInit_
{
  UsbCallback callback;
  unsigned char classCode;
  unsigned char subClassCode;
  unsigned char protocolCode;
  unsigned vendorId;
  unsigned productId;
  unsigned versionBcd;
  char far *manufacturerString;
  char far *productString;
  char far *serialNumberString;
  char far *configString;
  BOOL_ selfPowered;
  unsigned char maxPowerConsumption;
  unsigned char pioNumber;
} UsbInit;
```

## Comments

This structure is used to pass arguments to the interrupt service **USB_SERVICE_DEVICE_INIT**. It is not needed if you use the C-library function **usbDeviceInit()**.

**callback**

Pointer to the callback-function that the USB-driver will call to notifiy the application of certain events

**classCode**
Class code of the device

**subClassCode**
Subclass code of the device

**protocolCode**
Protocol code of the device

The Host uses classCode, subClassCode and protocolCode to find an appropriate standard driver for the device. If classCode is 0xFF - which means that the device uses a vendor-specific protocol - the host uses vendor and product ID to find a driver.
classCode, subClassCode and protocolCode may also be set with interfaces. In this case the values here should be 0x00.

**vendorId**
Vendor ID of the device. The vendor ID identifies the vendor of a device. It is assigned by the **USB Implementers Forum** .

**productId**
Product code of the device. The several devices of a vendor are differed by their product ID. The vendor is free to assign any product ID.

**versionBcd**
Release version of the device in BCD-format. This is used to distinguish several releases of the same product.

The following are human readable strings describing the device. The host may show these to the user. The maximum length of a string is USB_SPEC_MAX_STRING_LENGTH characters. If the string is longer it will be cut. Note that they have to be static, since the driver will not copy their content but only hold a pointer. If a string shall not be used, set the pointer to NULL.

**manufacturerString**
Pointer to manufacturer string (e.g. "Beck IPC GmbH")

**productString**
Pointer to product string (e.g. "IPC@CHIP SC123")

**serialNumberString**
Pointer to serial-number string (e.g. "123456-ABCDE")

**configString**
Pointer to configuration string

**selfPowered**
TRUE, if the device is self-powered

**maxPowerConsumption**
Maximum current that the device may draw from the bus. Given in steps of 2mA.

**pioNumber**
Number of PIO to be used to control attach- and speed-detection circuit; Refer to SC1x3 hardware manual.  On SC23 this circuit is integrated at the chip and hardwired to PIO 26.

## Related Topics

Interrupt service **USB_SERVICE_DEVICE_INIT**
Type **UsbCallback**

**Top of list**

# UsbDevice

```
typedef struct UsbDevice_
{
  unsigned int usbVersion;
  unsigned char classCode;
  unsigned char subClassCode;
  unsigned char protocolCode;
  unsigned char maxPacketSizeEp0;
  unsigned int vendorId;
  unsigned int productId;
  unsigned int version;
  unsigned char manufacturerStringIndex;
  unsigned char productStringIndex;
  unsigned char serialNumberStringIndex;
  unsigned char nConfiguration;
} UsbDevice;
```

## Comments

This structure represents the information from the Device Descriptor of a device. It is returned by the API function **usbHostGetDeviceInfo()** (Interrupt service **USB_SERVICE_HOST_GET_DEVICE_INFO**). For detailed information on the Device Descriptor refer to the USB specification.

**usbVersion**
USB specification version that the device supports, encoded in BCD

**classCode**
Class code of the device

**subClassCode**
Subclass code of the device

**protocolCode**
Protocol code of the device

Class code, Subclass code and Protocol code are used to find an appropriate standard driver for the device. If classCode is 0xFF - which means that the device uses a vendor-specific protocol - vendor and product ID are used to find a driver.
classCode, subClassCode and protocolCode may also be set with interfaces. In this case the values here will be 0x00.

**maxPacketSizeEp0**
Maximum packet size supported on endpoint 0

**vendorId**
Vendor ID of the device. The vendor ID identifies the vendor of a device. It is assigned by the **USB Implementers Forum** .

**productId**
Product code of the device. The several devices of a vendor are differed by their product ID. The vendor is free to assign any product ID.

**version**
Release version of the device in BCD-format.

If the Class code of the device is 0xFF, the device uses a vendor-specific protocol. Vendor and product ID as well as the product version are used to find an appropriate driver. Still it is possible that single interfaces are handled by a standard driver.

**`manufacturerStringIndex`, `productStringIndex` and `serialNumberStringIndex`**
These are the indexes of strings containing the name of the manufacturer, the name of the product and the serial number of the device. If an index is 0, this means that the resp. string is not available. Otherwise the function **usbHostGetString()**(Interrupt service **USB_SERVICE_HOST_GET_STRING** ) can be used to retrieve the string.

**`nConfigurations`**
The RTOS USB-driver doesn't support more than 1 Configuration. So this field will always have the value 1.

## Related Topics

API function **usbHostGetDeviceInfo()**
Interrupt service **USB_SERVICE_HOST_GET_DEVICE_INFO**
API function **usbHostGetString()**
Interrupt service **USB_SERVICE_HOST_GET_STRING**

**Top of list**
**Index page**

---

# UsbConfig

```
typedef struct UsbConfig_
{
  unsigned char nInterface;
  unsigned char stringIndex;
  BOOL_ selfPowered;
  BOOL_ remoteWakeup;
  unsigned char maxPowerConsumption;
  char *specificDesc;
  unsigned short specificDescLength;
} UsbConfig;
```

## Comments

This structure represents the information from the Configuration Descriptor of a device. It is returned by the API function **usbHostGetConfigInfo()** (Interrupt service **USB_SERVICE_HOST_GET_CONFIG_INFO**). For detailed information on the Configuration Descriptor refer to the USB specification.

**`nInterface`**
Number of Interfaces defined in this Configuration

**`stringIndex`**
Index of the string describing the Configuration - 0 if the string is not available. The function **usbHostGetString()** (Interrupt service **USB_SERVICE_HOST_GET_STRING**) can be used to retrieve the string.

**`selfPowered`**
TRUE, if the device is self-powered (i.e. it has its own power supply)

**remoteWakeup**
TRUE, if the device supports the Remote Wakeup Feature

**maxPowerConsumption**
Maximum power the device may consume in units of 2 mA. The RTOS USB-driver checkes whether it can supply the power that devices say they need. By default the driver assumes that 500 mA can be supplied. This value can be changed via an entry in the **CHIP.INI** configuration file.

**specificDesc**
Class- or vendor-specific descriptor(s) assigned to the Configuration. Don't manipulate this data.

**speceifcDescLength**
Length of the class- or vendor-specific descriptor data in bytes.

## Related Topics

API function **usbHostGetConfigInfo()**
Interrupt service **USB_SERVICE_HOST_GET_CONFIG_INFO**
API function **usbHostGetString()**
Interrupt service **USB_SERVICE_HOST_GET_STRING**

**Top of list**
**Index page**

---

# UsbInterface

```
typedef struct UsbInterface_
{
  unsigned char nAltSetting;
  unsigned char nEp;
  unsigned long epBitmap;
  unsigned char classCode;
  unsigned char subClassCode;
  unsigned char protocolCode;
  unsigned char stringIndex;
  char *specificDesc;
  unsigned short specificDescLength;
} UsbInterface;
```

## Comments

This structure represents the information from an Interface Descriptor. It is returned by the API function **usbHostGetInterfaceInfo()** (Interrupt service **USB_SERVICE_HOST_GET_INTERFACE_INFO**). For detailed information on the Interface Descriptor refer to the USB specification.

**nAltSetting**
Number of Alternate Settings the interface has

**nEp**
Number of Endpoints assigned to the Interface/Alternate Setting

**epBitmap**
Bitmap field representing the the endpoint numbers that are assigned to the interface. IN and

OUT endpoints are distinguished here. The lower 16 bit represent the OUT endpoints, the higher 16 bit represent the IN endpoints. If e.g. bit 3 ist set, endpoint 3 OUT is assigned to the interface. If bit 31 is set, endpoint 15 IN is assigned to the interface. For control endpoints both directions are assigned of course. So if the control endpoint 4 is assigned to the interface, the bits 4 and 20 are set.

**`classCode`**
Class code of the Interface

**`subClassCode`**
Subclass code of the Interface

**`protocolCode`**
Protocol code of the Interface

For details on these codes refer to the description of the structure **UsbDevice**.

**`stringIndex`**
Index of the string describing the Interface - 0 if the string is not available. The function **usbHostGetString()** (Interrupt service **USB_SERVICE_HOST_GET_STRING**) can be used to retrieve the string.

**`specificDesc`**
Class- or vendor-specific descriptor(s) assigned to the Interface. Don't manipulate this data.

**`speceifcDescLength`**
Length of the class- or vendor-specific descriptor data in bytes.

## Related Topics

API function **usbHostGetInterfaceInfo()**
Interrupt service **USB_SERVICE_HOST_GET_INTERFACE_INFO**
API function **usbHostGetString()**
Interrupt service **USB_SERVICE_HOST_GET_STRING**

**Top of list**
**Index page**

---

# UsbEp

```
typedef struct UsbEp_
{
  unsigned char transferType;
  unsigned int maxPacketSize;
  unsigned char interval;
  char *specificDesc;
  unsigned short specificDescLength;
} UsbEp;
```

## Comments

This structure represents the information from an Endpoint Descriptor. It is returned by the API function **usbHostGetEpInfo()** (Interrupt service **USB_SERVICE_HOST_GET_EP_INFO**). For detailed information on the Endpoint Descriptor refer to the USB specification.

**`transferType`**
Transfer type of the endpoint; possible values are:
- `USB_SPEC_TRANSFER_TYPE_CONTROL`
- `USB_SPEC_TRANSFER_TYPE_ISOCHRONOUS`
- `USB_SPEC_TRANSFER_TYPE_BULK`
- `USB_SPEC_TRANSFER_TYPE_INTERRUPT`

The symbolic names are defined in USB_SPEC.H.

**`maxPacketSize`**
The maximum packet size of this endpoint

**`interval`**
The interval at which this endpoint is polled by the Host if it is an INTERRUPT or ISOCHRONOUS endpoint. The interval is measured in 1-ms-frames.

**`specificDesc`**
Class- or vendor-specific descriptor(s) assigned to the Endpoint. Don't manipulate this data.

**`speceifcDescLength`**
Length of the class- or vendor-specific descriptor data in bytes.

## Related Topics

API function **usbHostGetEpInfo()**
Interrupt service **USB_SERVICE_HOST_GET_EP_INFO**

**Top of list**
**Index page**

---

End of document

---

# @CHIP-RTOS - USB Error Codes

## Error Codes

Several USB API functions return error codes. If a function call was successful a positiv value or 0 (= USB_ERROR_NO) is returned. A negativ result indicates an error.

**USB_ERROR_NO** = 0
: Function call was successful

**USB_ERROR_ILLEGAL_ARGUMENT** = -1
: One of the arguments passed to the function was illegal (e.g. out of range).

**USB_ERROR_OUT_OF_MEMORY** = -2
: The USB-driver could not allocate enough memory for the requested operation

**USB_ERROR_RESOURCE_LACK** = -3
: Lack of some kind of internally managed resource such as tasks, semaphores, buffers, etc.

**USB_ERROR_QUEUE_MAXIMUM** = -4
: A request could not be enqueued due to the resp. queue having reached its maximum.

**USB_ERROR_FEATURE_DISABLED** = -5
: The requested feature is not enabled. This error e.g. occures if you try to wakup the Host in Device mode when the Remote Wakeup Feature has been disabled by the Host previously.

**USB_ERROR_OUT_OF_INTERFACES** = -6
: There were not enough free interfaces/alternate settings left.

**USB_ERROR_OUT_OF_ENDPOINTS** = -7
: There were not enough free endpoints left.

**USB_ERROR_RESET_SETUP** = -8
: A reset occured or a SETUP token has been received on the resp. endpoint but the event has not yet been notified. The requested action cannot be performed in this situation. (Device mode only)

**USB_ERROR_UNKNOWN_SERVICE** = -9
: The software interrupt was triggered with an unknown service number.

**USB_ERROR_PROTOCOL** = -10
: The USB-Protocol was violated or the protocol indicated an error.

**USB_ERROR_BUSY** = -11
: A resource is temporarily not available but might be available again later

End of document

## usbSend

Enqueues a send request to an endpoint's TX queue. In Device mode the data will be transmitted when the Host issues an IN token. In Host mode the data will be transmitted when a SETUP or OUT token is issued using **usbHostIssueToken**. When the request has been completed the **callback**-function will be called with a status of USB_EVENT_SENT.

```
int usbSend( unsigned char address,
                    unsigned char epNumber,
                    void far *buffer,
                    unsigned length,
                    UsbBool less );
```

## Parameters

*address*

Host mode: Address of the device that is to receive the data
Device mode: Must always be 0x00

*epNumber*

Number of the enpoint via which the data is to be sent

*buffer*

Data to be sent

*length*

Length of data to be sent

*less*

Host mode: Must always be FALSE
Device mode: Set this parameter to TRUE if the host demanded more data than is to be sent. This does only apply to control endpoints.

## Return Value

**Error code**

## Comments

The buffer that is used for sending must be allocated by the application and be available while the USB-

driver processes the send-request. After it has completed the callback-function will be called with an USB_EVENT_SENT event (or maybe an error event). After that the buffer can be released.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_SEND** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbDeviceConfigEp**
C-library function **usbHostIssueToken**
C-library function **usbHostClearQueue**
C-library function **usbHostReserveEp0**
C-library function **usbHostReleaseEp0**
Type **UsbCallback**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbFreeReceiveBuffer

Frees a receive buffer. The RTOS USB-driver manages a bunch of receive buffers internally. Each time data is received on any endpoint, it is put into one of these buffers and the buffer is handed to a user driver along with a callback event.
To enable the reuse of the buffer, this function has to be called as soon as possible.

```
void usbFreeReceiveBuffer( void far *buffer );
```

## Parameters

*buffer*

Pointer to the receive buffer that is to be freed

## Return Value

None

## RTOS API

This library function uses a dynamic link to reach the function
USB_SERVICE_FREE_RECEIVE_BUFFER offered by RTOS USB Software Interrupt.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## usbWakeup

ends a resume signal to the bus to wakeup suspended nodes

```
int usbWakeup( void );
```

## Parameters

None

## Return Value

Error code

## Comments

As well hosts as devices may perform resume signaling. But the Remote Wakeup Feature of devices may be disabled. In this case a call to this function will fail in Device mode.

## RTOS API

This library function uses a dynamic link to reach the function USB_SERVICE_WAKEUP offered by RTOS USB Software Interrupt.

## Related Topics

C-library function usbHostSetClearFeature

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbSofNotify

Use this function to enable or disable the forwarding of SOF events to your application. By default the event **USB_EVENT_SOF** is not notified.

```
int usbSofNotify( unsigned char driverId,
                  UsbBool enable );
```

## Parameters

*driverId*

Host mode: ID of the user driver
Device mode: Must always be 0x00

*enable*

Indicates whether to enable (TRUE) or disable (FALSE) the notification of SOF events

## Return Value

**Error code**

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_SOF_NOTIFY** offered by RTOS USB Software Interrupt.

## Related Topics

Type **UsbCallback**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|

| V2.05 |
| --- |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbDeviceInit

Initialises the USB driver in Device mode and registers a **callback**-function that will be called to notify the user driver of events like completed transactions.
Also sets sundry strings that describe the device.
Only one user program can use the USB driver in Device mode.

```
int usbDeviceInit( UsbCallback callback,
                   unsigned char classCode,
                   unsigned char subClassCode,
                   unsigned char protocolCode,
                   unsigned vendorId,
                   unsigned productId,
                   unsigned versionBcd,
                   char far *manufacturerString,
                   char far *productString,
                   char far *serialNumberString,
                   char far *configString,
                   UsbBool selfPowered,
                   unsigned char maxPowerConsumption
                   unsigned char pioNumber );
```

## Parameters

*callback*

Pointer to the callback-function that the USB-driver will call to notifiy the application of certain events.

*classCode*

Class code of the device

*subClassCode*

Subclass code of the device

*protocolCode*

Protocol code of the device

*vendorId*

Vendor ID of the device. The vendor ID identifies the vendor of a device. It is assigned by the **USB Implementers Forum** .

*productId*

Product code of the device. The several devices of a vendor are differed by their product ID. The vendor is free to assign any product ID.

*versionBcd*

Release version of the device in BCD-format. This is used to distinguish several releases of the same product.

*manufacturerString*

Pointer to manufacturer string (e.g. "Beck IPC GmbH")

*productString*

Pointer to product string (e.g. "IPC@CHIP SC123")

*serialNumberString*

Pointer to serial-number string (e.g. "123456-ABCDE")

*configString*

Pointer to configuration string

*selfPowered*

TRUE, if the device is self-powered

*maxPowerConsumption*

Maximum current that the device may draw from the bus. Given in steps of 2mA.

*pioNumber*

Number of PIO to be used to control attach- and speed-detection circuit; Refer to SC1x3 hardware manual.  On SC23 this circuit is integrated at the chip and hardwired to PIO 26.

## Return Value

**Error code**

## Comments

The Host uses classCode, subClassCode and protocolCode to find an appropriate standard driver for the device. If classCode is 0xFF - which means that the device uses a vendor-specific protocol - the host uses vendor and product ID to find a driver.
classCode, subClassCode and protocolCode may also be set with interfaces. In this case the values here should be 0x00.

manufacturerString, productString, serialNumberString and configString are human readable strings describing the device. The host may show these to the user. The maximum length of a string is USB_SPEC_MAX_STRING_LENGTH characters. If the string is longer it will be cut. Note that they have to be static, since the driver will not copy their content but only hold a pointer. If a string shall not be used, set the pointer to NULL.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_DEVICE_INIT** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbDeviceDeinit**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbDeviceDeinit

This function deinitialises the USB-driver in Device mode.

```
void far usbDeviceDeinit( void );
```

## Parameters

None

## Return Value

None

## Comments

If the USB-driver has already been started, stop it first.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_DEVICE_DEINIT** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbDeviceInit**
C-library function **usbDeviceStop**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbDeviceRegisterInterface

This function registers a new interface.

```
int usbDeviceRegisterInterface( unsigned char nEp,
                    unsigned char classCode,
                    unsigned char subClassCode,
                    unsigned char protocolCode,
                    char far *interfaceString );
```

## Parameters

*nEp*

Number of endpoints to associate with the interface

*classCode*

Class code of this interface

*subClassCode*

Subclass code of this interface

*protocolCode*

Protocol code of this interface

*interfaceString*

Human readable string describing this interface, must be static for the USB-driver will not copy it. The string can have a maximum length of USB_SPEC_MAX_STRING_LENGTH characters. If no string is needed, set this to NULL.

## Return Value

ID of the newly registered interface or **Error code**

## Comments

For further information on Class, Subclass and Protocol code see the description of **usbDeviceInit**. Use the function **usbDeviceGetEps** to get the numbers of the endpoints that are assigned to the interface.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_DEVICE_REGISTER_INTERFACE** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbDeviceInit**
C-library function **usbDeviceRegisterAltSetting**
C-library function **usbDeviceGetEps**
C-library function **usbDeviceAddSpecificDesc**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## usbDeviceRegisterAltSetting

This function registers a new alterate setting to a given interface.

```
int usbDeviceRegisterAltSetting( unsigned char interfaceId );
```

## Parameters

*interfaceId*

ID of the interface to that the alternate settings is to be registered

## Return Value

ID of the new alternate setting or **Error code**

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_DEVICE_REGISTER_ALT_SETTING** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbDeviceRegisterInterface**
C-library function **usbDeviceConfigEp**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**

End of document

## usbDeviceGetEps

This function gets the numbers of the endpoints that have been associated with an interface. The numbers are returned in a bitmap field. Each bit that is set to "1" marks an assigned endpoint. If e.g. Bit 1 is set, endpoint 1 is assigned to the interface.
By default both the IN and OUT endpoint at the resp. endpoint number are reserved to the interface. You decide which direction actually to assign to the interface when configuring the endpoints. You may even use both directions if both have the same transfer type.

```
int usbDeviceGetEps( unsigned char interfaceId,
                     unsigned far *epBitmap );
```

## Parameters

*interfaceId*

ID of the interface whose endpoints are to be gotten

*epBitmap*

Result parameter: Bitmap indicating which ndpoints belong to the given interface.

## Return Value

Error code

## Comments

Endpoint 0 never can be assigned to an interface but every interface can use it.

## RTOS API

This library function uses a dynamic link to reach the function USB_SERVICE_DEVICE_GET_EPS offered by RTOS USB Software Interrupt.

## Related Topics

C-library function usbDeviceRegisterInterface
C-library function usbDeviceConfigEp

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbDeviceConfigEp

This function can be used to configure an endpoint. Transfer type and the transfer direction can be configured once for each endpoint. The maximum packet size and the polling interval can be configured for each alternate setting of the interface that the endpoint is associated to.

```
int usbDeviceConfigEp( unsigned char interfaceId,
                       unsigned char altSettingId,
                       unsigned char epNumber,
                       unsigned char transferType,
                       UsbBool in,
                       unsigned maxPacketSize,
                       unsigned char interval );
```

## Parameters

*interfaceId*

ID of the interface the endpoint is associated to

*altSettingId*

ID of the alternate setting to the given interface

*epNumber*

Number of the endpoint that is to be configured

*transferType*

Transfer type; possible values are:

- USB_SPEC_TRANSFER_TYPE_CONTROL
- USB_SPEC_TRANSFER_TYPE_ISOCHRONOUS
- USB_SPEC_TRANSFER_TYPE_BULK
- USB_SPEC_TRANSFER_TYPE_INTERRUPT

The symbolic names are defined in USB_SPEC.H.

*in*

TRUE, if this is an IN-endpoint, which means that data is transfered from the device to the host. This parameter is ignored for control-endpoints.

*maxPacketSize*

Maximum size of packets sent/received through this endpoint; Please note that the packet size may also be limited by the USB specification and by the capabilities of the controller.

*interval*

Polling interval for interrupt and isochronous endpoints; given in 2ms steps.

## Return Value

Maximum packet size set (respecting the USB specification and the capabilities of the controller) or **Error code**

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_DEVICE_CONFIG_EP** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbDeviceRegisterInterface**
C-library function **usbDeviceRegisterAltSetting**
C-library function **usbDeviceGetEps**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbDeviceAddSpecificDesc

This function can be used to add class- oder vendor-specific descriptors to the Configuration, to an interface or an endpoint. A buffer has to be allocated by the application into which the descriptor is written. Several descriptors can be written in this single buffer, but only one buffer can be added to each entity. The buffer must be available during the whole program execution because the RTOS USB-driver will not copy it. The specific descriptors will be concatenated to the resp. standard descriptors.

```
int usbDeviceAddSpecificDesc( unsigned char descriptorType,
                    unsigned char idNumber,
                    char far *descriptor,
                    unsigned char descriptorLength );
```

## Parameters

*descriptorType*

Type of entity to which the specific descriptor is to be added. These are possible values:

- USB_SPEC_DESCRIPTOR_TYPE_CONFIG
- USB_SPEC_DESCRIPTOR_TYPE_INTERFACE
- USB_SPEC_DESCRIPTOR_TYPE_EP

The symbolic names are defined in USB_SPEC.H.

*idNumber*

ID/number of entity to which the specific descriptor is to be added. If the descriptor type is USB_SPEC_DESCRIPTOR_TYPE_EP this is the endpoint number. If descriptor type is USB_SPEC_DESCRIPTOR_TYPE_INTERFACE this is the index of the interface. If descriptor type is USB_SPEC_DESCRIPTOR_TYPE_CONFIG this parameter should be 0.

*descriptor*

Pointer to the buffer containing the descriptor(s)

*descriptorLength*

Length of the descriptor(s) in bytes

## Return Value

Error code

## Comments

If a specific descriptor is added to an interface it is added to all alternate settings.

## RTOS API

This library function uses a dynamic link to reach the function
**USB_SERVICE_DEVICE_ADD_SPECIFIC_DESC** offered by RTOS USB Software Interrupt.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbDeviceStart

This function will start the Device mode USB-driver and enable the attach- and speed-detection circuit via the configured PIO pin.

```
void usbDeviceStart( void );
```

## Parameters

None

## Return Value

None

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_DEVICE_START** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbDeviceStop**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## usbDeviceStop

This function will stop the Device mode USB-driver and disable the attach- and speed-detection circuit via the configured PIO pin.

```
void usbDeviceStop( void );
```

## Parameters

None

## Return Value

None

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_DEVICE_STOP** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbDeviceStart**
C-library function **usbDeviceDeinit**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## usbDeviceStallEp

This function will set the STALL condition for an endpoint. The USB controller will reject all arriving packets and send requests on this endpoint and return a STALL handshake. A distinction between the IN and OUT direction is not possible. Both directions will be stalled.
Host intervention is needed to reset the condition.

```
int usbDeviceStallEp( unsigned char epNumber );
```

## Parameters

*epNumber*

Number of endpoint that is to be stalled

## Return Value

Error code

## RTOS API

This library function uses a dynamic link to reach the function USB_SERVICE_DEVICE_STALL_EP offered by RTOS USB Software Interrupt.

## Related Topics

C-library function usbHostSetClearFeature

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbDeviceEnableEp

This function enables or disables communication in the OUT direction (Host to Device) of an endpoint. When the Host tries to send data on a disabled endpoint a NAK handshake will be returned (except for ISOCHRONOUS endpoints).

By default all configured endpoints are enabled after the Device has been started.

Note that this function will not stop reception immediately. At least two more packets can be received, because respective requests have already been handed to the controller hardware. Some more data may already have been received, but still be pending in a software queue inside the RTOS.

```
int usbDeviceEnableEp( unsigned char epNumber
                       UsbBool enable );
```

## Parameters

*epNumber*

Number of endpoint that is to be enabled/disabled

*enable*

Indicates whether the endpoint is to be enabled (TRUE) or disabled (FALSE)

## Return Value

**Error code**

## Comments

If the device is permanent unable to receive data, the endopint should be stalled.

To achieve sending of a NAK handshake in the IN direction (Device to Host) of an endpoint, simply stop sending on this endpoint. If the Host then tries to receive data on that endpoint, a NAK handshake is returned automatically.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_DEVICE_ENABLE_EP** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbDeviceStallEp**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.20 | V1.20 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.13 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostInit

This function registers a user driver and initialises the RTOS USB-driver for Host mode if no other user driver has already done this. There can be several user drivers loaded in Host mode. Each driver registers a **callback**-function which is called when an event (e.g. completed transaction) appears on a device that the driver is assigned to. User drivers are assigned to Interfaces which allows a device with two different functions and thus two different Interfaces to be handled by two different drivers. The criteria for the assignment of user drivers to Interfaces are the codes handed to this function.

```
int usbHostInit( UsbCallback callback,
                 unsigned char classCode,
                 unsigned char subClassCode,
                 unsigned char protocolCode,
                 unsigned vendorId,
                 unsigned productId,
                 unsigned versionBcd );
```

## Parameters

*callback*

Callback function

*classCode*

Class code

*subClassCode*

Subclass code

*protocolCode*

Protocol code
If the user driver can handle devices of one of the Device Classes, these arguements identify the class, subclass and protocol. If the user driver handles vendor-specific protocols, set these arguments to 0.

*vendorId*

Vendor ID

*productId*

Product ID

*versionBcd*

Product version in BCD code
If the user driver handles devices using a vendor-specific protocol, these arguements identify the Vendor and Product ID of the resp. device as well as the maximum product version that the driver can handle. If the user driver can handle devices of one of the Device Classes, set these arguments to 0.

## Return Value

Assigned driver ID or **Error code**. The driver ID will be needed for other API functions.

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_INIT** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbHostDeinit**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## usbHostDeinit

This function unregisters a user driver and deinitialises the USB driver if there are no more user drivers registered.

```
void usbHostDeinit( unsigned char driverId );
```

## Parameters

*driverId*

ID of the user driver that is to be unregistered

## Return Value

None

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_DEINIT** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbHostInit**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostIssueToken

Issues an IN, OUT or SETUP token on a given endpoint. This is needed in Host mode to initiate a data transfer. To receive a packet from a device, an IN token has to be issued to tell the device to start sending. To send a packet to a device an OUT or SETUP token has to be issued after callingb **usbSend** to tell the RTOS USB-driver to start transmission.

```
int usbHostIssueToken( unsigned char address,
                       unsigned char epNumber,
                       unsigned char token );
```

## Parameters

*address*

Address of the device that is to receive the token

*epNumber*

Number of the endpoint that is to receive the token

*token*

Token that is to be sent; must be one of the following:

- USB_SPEC_PID_IN
- USB_SPEC_PID_OUT
- USB_SPEC_PID_SETUP

The symbolic names are defined in USB_SPEC.H.

## Return Value

**Error code**

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_ISSUE_TOKEN** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbSend**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostClearQueue

This function clears the send and token queue of a given endpoint. It can be used to reset an endpoint when an error occured.
Buffers associated with send requests cleared by this function will later be handed back to the user driver via an event with the status **USB_EVENT_RELEASE**.
Note that queue entries already passed to the USB controller cannot be removed.

```
int usbHostClearQueue( unsigned char address,
                       unsigned char epNumber,
                       UsbBool in );
```

## Parameters

*address*

Address of the resp. device

*epNumber*

Number of the endpoint whose queues are to be cleared

*in*

States that the IN direction of the endpoint is to be cleared; set to FALSE for OUT or CONTROL endpints

## Return Value

**Error code**

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_CLEAR_QUEUE** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbSend**
C-library function **usbHostIssueToken**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostReserveEp0

Reserves endpoint 0 of a device for a user driver. Since it might happen that several user drivers share one device, access to endpoint 0 of the device has to be synchronised. A user driver must not use (i.e. send or issue tokens) endpoint 0 before reserving it.
The endpoint has to be released again using the **usbHostReleaseEp0**.

```
int usbHostReserveEp0( unsigned char driverId,
                       unsigned char address );
```

## Parameters

*driverId*

ID of the user driver

*address*

Address of the resp. device

## Return Value

**Error code**

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_RESERVE_EP0** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbHostReleaseEp**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostReleaseEp0

Releases endpoint 0 of a device after it was reserved using **usbHostReserveEp0**.

```
int usbHostReleaseEp0( unsigned char driverId,
                       unsigned char address );
```

## Parameters

*driverId*

ID of the user driver that has reserved the endpoint

*address*

Address of the resp. device

## Return Value

**Error code**

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_RELEASE_EP0** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbHostReserveEp0**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostGetDeviceInfo

Returns information from the Device Descriptor of a given device

```
int usbHostGetDeviceInfo( unsigned char address,
                          UsbDevice far *info,
                          unsigned char size );
```

## Parameters

*address*

Address of the resp. device

*info*

Result parameter: Pointer to a structure of the type **UsbDevice** which is to be filled with the information

*size*

Size of the handed structure in bytes

## Return Value

**Error code**

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_GET_DEVICE_INFO** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbHostGetConfigInfo**
C-library function **usbHostGetInterfaceInfo**
C-library function **usbHostGetEpInfo**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostGetConfigInfo

Returns information from the Configuration Descriptor of a given device

```
int usbHostGetConfigInfo( unsigned char address,
                          UsbConfig far *info,
                          unsigned char size );
```

## Parameters

*address*

Address of the resp. device

*info*

Result parameter: Pointer to a structure of the type **UsbConfig** which is to be filled with the information

*size*

Size of the handed structure in bytes

## Return Value

**Error code**

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_GET_CONFIG_INFO** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbHostGetDeviceInfo**
C-library function **usbHostGetInterfaceInfo**
C-library function **usbHostGetEpInfo**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostGetInterfaceInfo

Returns information from the Interface Descriptor of a given interface

```
int usbHostGetInterfaceInfo( unsigned char address,
                     unsigned char interfaceId,
                     unsigned char altSettingId,
                     UsbInterface far *info,
                     unsigned char size );
```

## Parameters

*address*

Address of the resp. device

*interfaceId*

ID of the resp. Interface

*altSettingId*

ID of the resp. Alternate Setting

*info*

Result parameter: Pointer to a structure of the type **UsbInterface** which is to be filled with the information

*size*

Size of the handed structure in bytes

## Return Value

**Error code**

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_GET_INTERFACE_INFO** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbHostGetDeviceInfo**
C-library function **usbHostGetConfigInfo**
C-library function **usbHostGetEpInfo**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostGetEpInfo

Returns information from the Endpoint Descriptor of a given endpoint

```
int usbHostGetEpInfo( unsigned char address,
                      unsigned char altSettingId,
                      unsigned char epNumber,
                      UsbBool in,
                      UsbEp far *info,
                      unsigned char size );
```

## Parameters

*address*

Address of the resp. device

*altSettingId*

ID of the resp. alternate setting

*epNumber*

Number of the resp. endpoint

*in*

Indicates that information on an IN endpoint is desired; always set to FALSE for control-endpoints

*info*

Result parameter: Pointer to a structure of the type **UsbEp** which is to be filled with the information

*size*

Size of the handed structure in bytes

## Return Value

**Error code**

## Comments

This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_GET_EP_INFO** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbHostGetDeviceInfo**
C-library function **usbHostGetConfigInfo**
C-library function **usbHostGetInterfaceInfo**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostGetString

Retrieves a string from a given device

```
int usbHostGetString( unsigned char driverId,
                      unsigned char address,
                      unsigned char stringId,
                      char far *string );
```

## Parameters

*driverId*

ID of the user driver

*address*

Address of the resp. device

*stringId*

ID of the string that is desired

*string*

Pointer to a buffer where the string is to be stored; Must be large enough to store a string consisting of USB_SPEC_MAX_STRING_LENGTH characters plus trailing zero-byte

## Return Value

Error code

## Comments

This function reserves endpoint 0 of the resp. device for the user driver that calls it before using the endpoint and it releases it again afterwards.
Note that this function may take up to 2000 ms to complete, because it waits that long for data to arrive from the device.
This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function USB_SERVICE_HOST_GET_STRING offered by RTOS USB Software Interrupt.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostGetStatus

Retrieves status information from a given device using the GetStatus Standard Device Request

```
int usbHostGetStatus( unsigned char driverId,
                      unsigned char address,
                      unsigned char receiver,
                      unsigned char index,
                      unsigned short far *status );
```

## Parameters

*driverId*

ID of the user driver

*address*

Address of the resp. device

*receiver*

Receiver of the GetStatus request; this parameter decides whether the status of the device, an interface, an endpoint or a user- defined structure is to be retrieved; must be one of the following:

- USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_DEVICE
- USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_INTERFACE
- USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_ENDPOINT
- USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_OTHER

The symbolic names are defined in USB_SPEC.H.

*index*

Index of the resp. structure. The value of this parameter depends on the value of the receiver parameter.
If the receiver is USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_DEVICE, the index must always be 0.
If the receiver is USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_INTERFACE, the index must contain the ID of the resp. interface.
If the receiver is USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_ENDPOINT, the index parameter must contain the number of the resp. endpoint. If it is an IN endpoint, bit 7 has to be set additionally. (e.g. IN endpoint 1 => 0x81).
If the receiver is USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_OTHER, the value of the index parameter is device-specific.

*status*

Pointer to an `unsigned short` variable where the status is to be stored.

## Return Value

**Error code**

## Comments

This function reserves endpoint 0 of the resp. device for the user driver that calls it before using the endpoint and it releases it again afterwards.
Note that this function may take up to 2000 ms to complete, because it waits that long for data to arrive from the device.
This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_GET_STATUS** offered by RTOS USB Software Interrupt.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## usbHostSetClearFeature

Sets or clears a feature of a given device by sending a SetFeature or ClearFeature Standard Device Request

```
int usbHostSetClearFeature( unsigned char driverId,
                    unsigned char address,
                    unsigned char receiver,
                    unsigned char index,
                    unsigned char feature,
                    UsbBool setFeature );
```

## Parameters

*driverId*

ID of the user driver

*address*

Address of the resp. device

*receiver*

Receiver of the SetFeature or GetFeature request; this parameter decides whether a feature of the device, an interface, an endpoint or a user- defined structure is to be set/cleared; must be one of the following:

- USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_DEVICE
- USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_INTERFACE
- USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_ENDPOINT
- USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_OTHER

The symbolic names are defined in USB_SPEC.H.

*index*

Index of the resp. structure. The value of this parameter depends on the value of the receiver parameter.
If the receiver is USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_DEVICE, the index must always be 0.
If the receiver is USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_INTERFACE, the index must contain the ID of the resp. interface.
If the receiver is USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_ENDPOINT, the index parameter must contain the number of the resp. endpoint. If it is an IN endpoint, bit 7 has to be set additionally. (e.g. IN endpoint 1 => 0x81).
If the receiver is USB_SPEC_DEVICE_REQUEST_TYPE_RECEIVER_OTHER, the value of the index parameter is device-specific.

*feature*

Identifies the feature that is to be set/cleared. Actually the feature USB_SPEC_FEATURE_REMOTE_WAKEUP is defined for devices and USB_SPEC_FEATURE_STALL for endpoints.
The symbolic names are defined in USB_SPEC.H.

*setFeature*

If this parameter is TRUE the feature will be set, otherwise it will be cleared.

## Return Value

**Error code**

## Comments

This function reserves endpoint 0 of the resp. device for the user driver that calls it before using the endpoint and it releases it again afterwards.
Note that this function may take up to 2000 ms to complete, because it waits that long for data to arrive from the device.
This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function **USB_SERVICE_HOST_SET_CLEAR_FEATURE** offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbWakeup**
C-library function **usbDeviceStallEp**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## usbHostSetInterface

Sets one of the alternate settings of a given interface as the current one by sending a SetInterface Standard Device Request

```
int usbHostSetInterface( unsigned char driverId,
                         unsigned char address,
                         unsigned char interfaceId,
                         unsigned char altSettingId );
```

## Parameters

*driverId*

ID of the user driver

*address*

Address of the resp. device

*interfaceId*

ID of the resp. interface

*altSettingId*

ID of the alternate setting that is to be activated

## Return Value

Error code

## Comments

This function reserves endpoint 0 of the resp. device for the user driver that calls it before using the endpoint and it releases it again afterwards.
Note that this function may take up to 2000 ms to complete, because it waits that long for data to arrive from the device.
This function must not be called from within the callback-function.

## RTOS API

This library function uses a dynamic link to reach the function USB_SERVICE_HOST_SET_INTERFACE offered by RTOS USB Software Interrupt.

## Related Topics

C-library function **usbHostGetConfigInfo**
C-library function **usbHostGetInterfaceInfo**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS - CAN Bus Driver API



## CAN Bus Driver

The assembly language API definition for sending and receiving data on the CAN (Controller Area Network) buses is defined here. **This API is only available in the SC1x3/SC23 @CHIP-RTOS.**

The SC1x3/SC23 offers two CAN bus channels which can be operated with this API.

Each CAN channel has three hardware transmit registers and a four CAN message deep hardware receiver FIFO. The CAN driver provides send and receive software FIFO's with user specified sizes around each of these hardware devices.

One required step in using the CAN ports is to call the **CAN_Open_Port()** function, which configures the port for a specific baud rate and sets the driver's software FIFO sizes.

In hardware, each receive channel offers three configurable receiver filters which can be used to reduce the load placed on the CPU by the CAN receivers. These registers are configured with the **CAN_Rx_Filters()** function.

Use of the CAN API C library is recommended, rather than dealing directly with these low level assembly lanaguage interfaces specified here.

## Topics

CAN API **Data Structures**
CAN API **Error Codes**

## Interrupt 0xBF CAN Services

Service is selected by index in AH register.

- **0x80:_Install dynamic link, then call function**
- **0x81:_CAN_API_OPEN, Open a specified CAN port**
- **0x82:_CAN_API_RX_FILT, Initialize receiver filter**
- **0x83:_CAN_API_SEND, Send a message**
- **0x84:_CAN_API_RCV, Receive a message**
- **0x85:_CAN_API_PEEK, Look at message**
- **0x86:_CAN_API_RECONFIG, Reconfigure CAN port**
- **0x87:_CAN_API_STATUS, Check port status**
- **0x88:_CAN_API_CONTROL, Port controls**
- **0x89:_CAN_API_CLOSE, Close a CAN port**
- **0x8A:_CAN_API_EVENT, Wait on CAN event**
- **0x8B:_CAN_API_CALLBACK, Install a ISR callback**
- **0x8C:_CAN_API_ERRORS, Check port error counts**

## Interrupt 0xBF service 0x80: Install dynamic link, then call function

This function installs a far JMP opcode in the vicinity of the return instruction pointer to enter the specified CAN API function directly from the user code. This function is provided for efficiency in real-time applications.

## Parameters

*AH*

0x80

*AL*

Dynamic link index for function whose link is desired from following list:

| Index | CAN Function | |
|---|---|---|
| 0 | **CAN_Open_Port** | Open a CAN bus port |
| 1 | **CAN_Rx_Filters** | Control CAN receiver filters |
| 2 | **CAN_Send** | Send CAN message |
| 3 | **CAN_Recv** | Pick up received CAN message |
| 4 | **CAN_Peek** | Pick up CAN message and leave it in queue |
| 5 | **CAN_Reconfigure** | Reconfigure an open CAN bus port |
| 6 | **CAN_Status** | Get CAN port status |
| 7 | **CAN_Control** | Control FIFO's and device enables |
| 8 | **CAN_Close_Port** | Close a CAN bus port |
| 9 | **CAN_Event_Sleep** | Await event on either CAN port |
| 10 | **CAN_Set_Callback** | Install/Remove CAN ISR callback vector |
| 11 | **CAN_Error_Counts** | Read out Tx/Rx error counts |

## Return Value

Instruction Pointer is moved back 10 bytes so that newly installed far JMP will execute immediately.

(Refer to respective linked function's documention for further return value information.)

Note: All dynamically linked functions are free to use all registers except for DS, SI, DI, and BP registers which are preserved. This matches the Paradigm C, Borland C 5.02 and Microsoft Visual C 1.52 calling conventions.

## Comments

This API service is intended to be used by the RTOS C-Library exclusively. It is not recommended that it be used directly from within user applications.

A JMP FAR instruction will be written 10 bytes before the user's return instruction pointer. This far jump will transfer control directly into the RTOS from the user code when executed. This new far JMP will execute immediately upon return from this software interrupt due to the instruction pointer adjustment performed within the call.

Interrupts are masked and restored to their state on entry.

Since the code at calling site will be patched by this RTOS call, it is important that it has exactly the following form (the link index carried in AL register will vary)

```
JMP SHORT over_patch  ; Occupies 2 bytes of instruction code here
NOP    ; Never executed (1 byte)
NOP    ; Never executed (1 byte)
NOP    ; never executed (1 byte)
over_patch:  ; Linker overwrites above 5 bytes with JMP FAR instruction
MOV AX,08000h  ; AH = 0x80, AL = 0 = index for CAN_Open_Port()
INT 0BFh
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x81: CAN_API_OPEN, Open a specified CAN port

Initializes a CAN port in preparation for use.

## Parameters

*AH*

0x81 (= CAN_API_OPEN)

*DX*

CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*ES:BX*

Far pointer to a **CAN_PORT_INIT** data structure set by user prior to calling. The CAN driver treats this data as read only.

## Return Value

AX = 0 success
AX non-zero = **error** code

## Comments

Either this function or the **CAN_Rx_Filters** (required in order to receive any CAN messages) would be the starting point in preparing a CAN port for use. This function allocates receive and send FIFOs. The port is initialized for operation at the specified **baud** rate.

An RTX **event** group is allocated if this action has not already been taken for the adjacent CAN port. (The driver uses a single RTX event group for the CAN port pair.)

No operation is performed if the port has already been opened. In this case an error code `CAN_EC_PORT_ALREADY_OPENED` is returned. If necessary, limited configuration changes on an open CAN port can be made with the **CAN_Reconfig** API.
Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

When CAN1 is opened, the PIO14 and PIO15 are configured for NORMAL operation.

**Important Note:**

No CAN messages will be received until the port's receiver filters have been enabled for reception using the CAN_Rx_FiltersAPI. Following a system reset, the CAN receiver filters are initialized to reject all incoming CAN messages by default.

## Beck C-Library Usage

CAN_Open_Port() - Open one of the CAN ports

## Related Topics

CAN_Rx_Filters - Initialize receiver filter
CAN_Close_Port - Close specified port
CAN_Reconfig - Adjust configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

Top of list
Index page

---

## Interrupt 0xBF service 0x82: CAN_API_RX_FILT, Initialize receiver filter

Initializes a set of receiver acceptance filter registers.

## Parameters

*AH*
0x82 (= CAN_API_RX_FILT)

*DX*
CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*CX*
Filter index: 0, 1 or 2

*ES:BX*
Far pointer to a CAN_RX_FILT data structure set by user prior to calling. The CAN driver treats this data as read only. Set to NULL to disable respective receiver filter.

## Return Value

AX = 0 success
AX non-zero = error code

## Comments

***This function must be called*** in order to receive any CAN messages on respective port. If all three message filters are disabled, no messages are received. This is the default condition following a reset.

This function is used to initialize one of the CAN receiver's three sets of acceptance filter registers. The

receiver acceptance filters may be used to limit the incoming message traffic to those CAN messages which apply to this node in the network.

To receive all messages, one message filter must be enabled and programmed with all its fields as "don't care" ('1's).

This function may be used either before or after the respective CAN port has been **opened**.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## Beck C-Library Usage

**CAN_Rx_Filters()** - Initialize a receiver filter

## Related Topics

**CAN_Open_Port** - Initialize CAN port

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x83: CAN_API_SEND, Send a message

Queue a CAN message for output on specified transmitter.

## Parameters

*AH*

0x83 (= CAN_API_SEND)

*DX*

CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*CX*

Transmit channel:
0 for Tx #1
1 for Tx #2
2 for Tx #3

or when operating in **CAN_TX_PRIORITY_QUEUES** mode:
0 for output through the normal priority send queue
1 for output through the high priority send queue
(2 works same as 1 here)

*ES:BX*

Far pointer to a **CAN_MSG** data structure set by user prior to calling. The CAN driver treats this data as read only.

## Return Value

AX = 0 success
AX non-zero = **error** code

## Comments

This API is used to send CAN messages. Refer to the C-Library **CAN_Send()** API documentation for a detailed explanation of this API's various modes of operation.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## Beck C-Library Usage

**CAN_Send()** - Transmit a CAN message

## Related Topics

**CAN_Recv** - Get a received CAN message
**CAN_Control** - Enable/disable transmitter

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x84: CAN_API_RCV, Receive a message

Pick up a received CAN message from receiver queue, removing the message from the queue.

## Parameters

*AH*
0x84 (= CAN_API_RCV)

*DX*
CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*CX*
Time-out
Negative value: No wait, return immediately
Zero: Wait forever for a CAN message
Positive value: Maximum milliseconds to wait for a CAN message

*ES:BX*
(Output Parameter) Far pointer to a **CAN_MSG** data structure where the driver will return the received CAN message.

## Return Value

AX = 0 success
AX non-zero = **error** code

## Comments

This function is used to pick up the next received CAN message from the receiver FIFO **queue**. The time-out parameter in CX register allows the caller to optionally sleep on the receiver a specified number of milliseconds until a received CAN message is available.

Note that if the CAN port is closed, the **CAN_Close_Port** API strobes the event bits to assure that a program thread which is sleeping on the receiver within this API will return.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

**Important Note:**

No CAN messages will be received until the port's receiver filters have been enabled for reception using the **CAN_Rx_Filters**API. Following a system reset, the CAN receiver filters are initialized to reject all incoming CAN messages by default.

## Beck C-Library Usage

**CAN_Recv()** - Receive a CAN message

## Related Topics

**CAN_Peek** - Read out a message, leave it in queue
**CAN_Rx_Filters** - Initialize CAN receiver filters
**CAN_Send** - Send a CAN message
**CAN_Close_Port** - Close a CAN port

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x85: CAN_API_PEEK, Look at message

Pick up a received CAN message from receiver queue, but leave the message in the queue.

## Parameters

*AH*
0x85 (= CAN_API_PEEK)

*DX*
CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*ES:BX*

(Output Parameter) Far pointer to a **CAN_MSG** data structure where the driver will return the received CAN message.

## Return Value

AX = 0 success
AX non-zero = **error** code

## Comments

This function picks up a copy of the next received CAN message from the receiver queue FIFO without removing that message from the queue.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

**Important Note:**

No CAN messages will be received until the port's receiver filters have been enabled for reception using the **CAN_Rx_Filters**API. Following a system reset, the CAN receiver filters are initialized to reject all incoming CAN messages by default.

## Beck C-Library Usage

**CAN_Peek()** - Look at next CAN message

## Related Topics

**CAN_Recv** - Get a received message
**CAN_Rx_Filters** - Initialize CAN receiver filters

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x86: CAN_API_RECONFIG, Reconfigure CAN port

Adjust baud rate or bit timing parameters on an open CAN port.

## Parameters

*AH*
0x86 (= CAN_API_RECONFIG)

*DX*
CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*ES:BX*
Far pointer to a **CAN_CONFIG** data structure. The driver treats this data structure in a read only

manner.

## Return Value

AX = 0 success
AX non-zero = **error** code

## Comments

This function allows bit timing to be adjusted on a CAN port which has already been opened. (Under normal circumstances it is not expected that this function will be used.)

This function may also be used to manually switch back on a CAN bus following a **bus off** event. By default, this is performed automatically by the driver. However the user may specify a manual **restart** mode.

A successful call to this API zeroes the port's CANn_EV_BUS_OFF event.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## Beck C-Library Usage

**CAN_Reconfig()** - Adjust baud rate or bit timing parameters

## Related Topics

**CAN_Open_Port** - Open a CAN port
**CAN_Rx_Filters** - Configure receiver filters
**CAN_Close_Port** - Close a CAN port
**CAN_Event** - Wait on CAN event

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x87: CAN_API_STATUS, Check port status

Read out status information for specified CAN port.

## Parameters

*AH*
0x87 (= CAN_API_STATUS)

*DX*
CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*ES:BX*

(Output Parameter) Far pointer to a **CAN_STATUS** data structure. The driver writes the status information into this data structure.

## Return Value

AX = 0 success
AX non-zero = **error** code

## Comments

This function reports various status information concerning the specified CAN port. After reading out the reported error flags, the internal error flag registers are cleared (destructive read).

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## Beck C-Library Usage

**CAN_Status()** - Read out status information

## Related Topics

**CAN_Error_Counts** - CAN port error counts
**CAN_Control** - CAN port control

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x88: CAN_API_CONTROL, Port controls

Enable or disable specified channels. Purge specified FIFO's.

## Parameters

*AH*
0x88 (= CAN_API_CONTROL)

*DX*
CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*BX*
"Purge FIFO's" bit field

*CX*
"Enable Device" bit field

*SI*

"Disable Device" bit field

## Return Value

AX = 0 success
AX non-zero = **error** code

## Comments

This function provides a way to enable or disable the receiver and the three individual transmit register channels. Also, any or all of the receiver or transmit FIFO's can be cleared.

The **CAN_Open_Port** API enables all three transmitters and optionally the receiver as well. So this provisional API here is not necessarily required for CAN port operation.

The input parameters in the `BX`, `CX`, and `SI` registers are each bit fields with the identical bit mapping as follows:

```
CAN_RX_SEL  = 0x8000, Receiver control flag
CAN_TX1_SEL= 0x0800, TX1 register control flag
CAN_TX2_SEL= 0x1000, TX2 register control flag
CAN_TX3_SEL= 0x2000, TX3 register control flag
```

A bit set indicates that the action associated with the particular CPU register should be taken for the device designed by the bit. For example if:

```
BX= CAN_RX_SEL| CAN_TX2_SEL;
   ... and ...
CX= CAN_TX3_SEL;
   ... and ...
SI= CAN_TX2_SEL;
```

then this API function will purge (delete its contents) the receiver FIFO and the FIFO associated with transmit register #2. It will enable transmit register #3. It will disable transmit register #2.

When transmit FIFO's are purged, any CAN message pending output in the specified hardware transmit register is also aborted.

In the non-sense case where a device's bit is set in both `CX` and `SI` registers, indicating that both an enabling and disabling action should be taken on the same device, the enabling will take precedence and the device will be enabled.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## Beck C-Library Usage

**CAN_Control()** - Enable or disable specified channels

## Related Topics

**CAN_Open_Port** - Configure CAN port
**CAN_Status** - Read CAN port status

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Interrupt 0xBF service 0x89: CAN_API_CLOSE, Close a CAN port

Close down a specified CAN port.

### Parameters

*AH*
0x89 (= CAN_API_CLOSE)

*DX*
CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

### Return Value

AX = 0 success
AX non-zero = **error** code

### Comments

This function closes a CAN port, releasing any resources it was using (e.g. heap memory for FIFO's and RTX event group). Prior to releasing the event group, the event bits in the RTX event group associated with this port are **strobed** in order to release any program threads which were waiting on events for this port which is now being closed.

Any user **callback** vectors that had been set for this port will be removed.

The CAN device is deactivated. Any pending transmission is aborted. Aborting a pending transmission requires some additional time beyond when this API returns such that if the port were to be immediately reopened, a subsequent call to **CAN_Send** could fail due to a transmit buffer still busy condition.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

### Beck C-Library Usage

**CAN_Close_Port()** - Close a CAN port

### Related Topics

**CAN_Port_Open** - Open a CAN port
**CAN_Recv** - Get a received message
**CAN_Event_Sleep** - Sleep on CAN event

### Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Interrupt 0xBF service 0x8A: CAN_API_EVENT, Wait on CAN event

Sleep for specified amount of time, awaiting any CAN event among those specified.

## Parameters

*AH*

0x8A (= CAN_API_EVENT)

*DX*

Set of CAN events to wait for (bit field)

*CX*

Time-out

Negative value: No wait, return immediately
Zero: Wait forever for the CAN event(s)
Positive value: Maximum milliseconds to wait for the CAN event(s)

## Return Value

AX = 0 indicates no events occurred (time-out)
AX non-zero = event bits which came true among those that were specified in DX.

## Comments

This function sleeps on the **event** group used internally by the CAN driver, awaiting specified CAN events. Note that the events can be on either of the two CAN ports.

The events specified in DX register can be any combination of the following events:

CAN0_EV_RX_RDY= 0x0001, CAN0 Receiver data ready
CAN0_EV_TX1_EMP= 0x0002, CAN0 Transmitter #1 queue empty
CAN0_EV_TX2_EMP= 0x0004, CAN0 Transmitter #2 queue empty
CAN0_EV_TX3_EMP= 0x0008, CAN0 Transmitter #3 queue empty
CAN0_EV_BUS_OFF= 0x0010, CAN0 Bus off condition

CAN1_EV_RX_RDY= 0x0100, CAN1 Receiver data ready
CAN1_EV_TX1_EMP= 0x0200, CAN1 Transmitter #1 queue empty
CAN1_EV_TX2_EMP= 0x0400, CAN1 Transmitter #2 queue empty
CAN1_EV_TX3_EMP= 0x0800, CAN1 Transmitter #3 queue empty
CAN1_EV_BUS_OFF= 0x1000, CAN1 Bus off condition

This function returns when any one of the specified events occurs. If no events are specified the function returns immediately with zero in AX. Any event bits in DX which apply to a port which is not open will be ignored and treated as zeros.

This function has no affect on the event flags. Event flags are ***not*** cleared as a side affect of this function call.

The CANn_EV_RX_RDY event is cleared after the respective receiver queue is emptied by calls to the **CAN_Recv** API. The CANn_EV_BUS_OFF event is cleared by calls to **CAN_Open_Port**, **CAN_Reconfig** or **CAN_Close_Port** API.

The transmit event flags operate differently depending on whether or not the CAN_TXQ1_DRIVES_ALL or CAN_TX_PRIORITY_QUEUES modes are being used. When either of these transmit modes are used, the **CAN_Send** API clears the transmit flags for all channels associated with the software queue that the output CAN message is directed to. And this same group of transmit channel event flags will be set to '1'

simulataneously when the software queue(s) feeding those channels is (are) emptied. However there may still be a CAN message or two awaiting output in the CAN hardware transmit registers after the events are set.

When the transmitters are operated in neither `CAN_TXQ1_DRIVES_ALL` nor `CAN_TX_PRIORITY_QUEUES` mode, the **CAN_Send** API clears the single event flag bit associated with the channel that the CAN message is directed. Then the respective transmit event bit will be set to '1' after the hardware register is emptied (CAN message sent) and no further queued messages were pending for this transmit device.

A port closing will cause this function to return if any event wait bits applied to that port which closed. In this case the event flags returned for the closed port should be ignored, since they were created artificially.

The handle to the event group which the CAN driver uses to implement this API can be retrieved with the **RTX_Find_EventGroup()** API provided that the user has not stepped on the name given this group with yet another event group by the same name. The name given this event group by the CAN driver is "_CAN". This event group exists only so long as one of the two CAN ports is **opened**.

An event group provides 16 independent events, of which this CAN driver uses ten (five events for each of two CAN ports). These ten event bits are specified above. The other six event flags are available to the user. If the user chooses to use these additional event flags for their own purposes then they should call directly the **RTX_Wait_For_Event()** rather then using this API. When this `CAN_API_EVENT` API is used, any event flags outside the scope of the ten used by the CAN driver are ignored. Furthermore, this API only waits for event flag '1' conditions. Using the `RTX_Wait_For_Event()` API directly, the user would have more flexibility.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## Beck C-Library Usage

**CAN_Event_Sleep()** - Sleep on event

## Related Topics

**CAN_Open_Port** - Open a CAN port
**CAN_Close_Port** - Close a CAN port
**CAN_Recv** - Receive CAN message
**CAN_Send** - Send CAN message
**CAN_Set_Callback** - Set ISR callback
**CAN_Reconfig** - Reconfigure port

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x8B: CAN_API_CALLBACK, Install a ISR callback

Close down a specified CAN port.

## Parameters

*AH*

0x8B (= CAN_API_CALLBACK)

*DX*

CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*ES:BX*

Vector to user's callback procedure

*CX*

Events bit field

## Return Value

AX = 0 success
AX non-zero = **error** code

## Comments

This function either installs or removes a user callback procedure that is called from within the CAN driver's Interrupt Service Routine (ISR) upon specified events. The vector in ES:BX registers can be set to NULL to remove existing vectors. Note that the **CAN_Close_Port** will remove any callbacks established for the CAN port being closed.

The callback will be made from within the CAN driver's interrupt service routine, which has been **registered** with the RTX @Chip-RTOS.

The system maintains three independent callbacks for each CAN port:

1. Receiver event callback: `CAN_CALLBACK_RX`
2. Transmit event callback: `CAN_CALLBACK_TX`
3. Bus off error event callback: `CAN_CALLBACK_ERR`

The `events` argument in CX register determines which callbacks are set by this API. The masks defined for this purpose in CAN_API.H are indicated above. Combinations of these masks can be formed by OR'ing them together when it is desired that events share the same user callback. For example, the `CAN_CALLBACK_ALL` mask will set all three callbacks to the single provided vector.

The set of three callbacks was implemented so that less logic is required inside the user callback to figure out why it had been called, when independent callback functions are used. Note that all three transmit events are mapped to a single transmit callback.

The prototype for the user callback is as follows.

```
void huge My_Callback(unsigned int event,
                      const CAN_MSG far *msg) ;
```

The `event` parameter in this callback will be a single bit that indicates the reason for the callback. This bit is the same as used for the **CAN_Event_Sleep** API's event masks, `CAN0_EV_RX_RDY` through `CAN1_EV_BUS_OFF`. Note that which CAN channel the event occurred on is conveyed by this event bit.

The `msg` pointer argument passed to the bus error callback is always NULL. No CAN message is associated with this callback. For receive events, the `msg` pointer references the CAN message that was just received. For transmit events, the `msg` pointer references the CAN message that just completed transmission.

The user callback function must preserve the DS, SS and SP registers. All other registers the callback is free to modify. The code within this callback must be keep to a minimum. This callback is performed within the CAN interrupt service. Interrupts are masked on entry to the callback and must remain masked. To encourage assembly language programming of this time critical callback, the parameters

are also made available in registers:

- CX - Contains the `event` parameter.
- ES:DI - Contains the `msg` far pointer parameter

These parameters are also pushed onto the stack to support a standard C language interface.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

**Note:**
Use of callbacks may overload the system when CPU is operated in **power save mode**and CAN is operated at high baud rates.

## Beck C-Library Usage

**CAN_Set_Callback()** - Set callback

## Related Topics

**CAN_Close_Port** - Close a CAN port
**CAN_Port_Open** - Open a CAN port
**CAN_Recv** - Get a received message
**CAN_Event_Sleep** - Sleep on CAN event

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

**Top of list**
**Index page**

---

## Interrupt 0xBF service 0x8C: CAN_API_ERRORS, Check port error counts

Read out the transmitter and receiver error counts and bus error status for a specified CAN port.

## Parameters

*AH*
0x8C (= CAN_API_ERRORS)

*DX*
CAN port specifier: 0 for CAN0 port, or 1 for CAN1 port

*ES:BX*
(Output Parameter) Far pointer to a **CAN_ERRORS** data structure. The driver writes the error status information into this data structure.

## Return Value

AX = 0 success
AX non-zero = **error** code

## Comments

This function reports transmitter/receiver error counts as specified by the Bosch CAN specification. Also some discrete bits are reported which indicate the bus error state.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## Beck C-Library Usage

**CAN_Error_Counts()** - Read out error counts

## Related Topics

**CAN_Status** - CAN port status
**CAN_Control** - CAN port control

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

**Top of list**
**Index page**

End of document

## CAN_Open_Port

Open one of the CAN ports.

```
int CAN_Open_Port ( unsigned int port_idx,
                    const CAN_PORT_INIT far *init );
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*init*

Pointer to port initialization data **structure** set by caller.

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success, CAN port opened.
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` invalid or NULL `init` pointer
`CAN_EC_INVALID_BAUD` = -2: Invalid baud rate or bit timing specification.
`CAN_EC_PORT_ALREADY_OPENED` = -4: Specified port was already opened
`CAN_EC_LOW_RESOURCES` = -7: Unable to create required event group.
`CAN_EC_INVALID_Q_SIZE` = -9: One of the queue sizes was invalid.
`CAN_EC_LOW_MEMORY` = -10: Unable to allocate heap memory.

## Comments

Either this function or the **CAN_Rx_Filters** (required in order to receive any CAN messages) would be the starting point in preparing a CAN port for use. This function allocates receive and send FIFOs per the caller's **FIFO size** specifications. The CAN port is initialized for operation at the specified **baud** rate. An RTX **event group** is allocated if this action has not already been taken for the adjacent CAN port. (The driver uses a single RTX event group for the CAN port pair.)

No operation is performed if the port has already been opened. In this case an error code `CAN_EC_PORT_ALREADY_OPENED` is returned. If necessary, limited configuration changes on an open CAN port can be made with the **CAN_Reconfig** API.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

When CAN1 is opened, the PIO14 and PIO15 are configured for NORMAL operation.

**Important Note:**

No CAN messages will be received until the port's receiver filters have been enabled for reception using the **CAN_Rx_Filters**API. Following a system reset, the CAN receiver filters are initialized to reject all incoming CAN messages by default.

## See Also

**CAN_Rx_Filters** **CAN_Close_Port** **CAN_Reconfig**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Rx_Filters

Initialize a receiver filter.

```
int CAN_Rx_Filters ( unsigned int port_idx,
                     unsigned int filter_idx,
                     const CAN_RX_FILT far *filt);
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*filter_idx*

Receiver filter specifier:
`CAN_FILT1`= 0
`CAN_FILT2`= 1
`CAN_FILT3`= 2

*filt*

Pointer to **filter** initialization data structure set by caller. This may be set to NULL to disable the respective receiver filter so that no CAN messages are accepted by the filter.

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` or `filter_idx` invalid

## Comments

***This function must be called*** in order to receive any CAN messages on respective port. If all three message filters are disabled, no messages are received. This is the default condition following a reset.

This function is used to initialize one of the CAN receiver's three sets of acceptance filter registers. The receiver acceptance filters may be used to limit the incoming message traffic to those CAN messages which apply to this node in the network.

To receive all messages, one message filter must be enabled and programmed with all its fields as "don't care" ('1's).

This function may be used either before or after the respective CAN port has been **opened**.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## See Also

**CAN_Open_Port**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Data Structures

Here are the data structures used by the CAN API. All constants and data structures referred to here are defined in the C-library header file CAN_API.H

Notes:

- **Byte alignment is required** for all data structures used within the API.

### Component List


- CAN_BOOL
- CAN_CONFIG
- CAN_ERRORS
- CAN_ID
- CAN_MSG
- CAN_PORT_INIT
- CAN_RX_FILT
- CAN_STATUS


---

# CAN_BOOL

```
typedef int CAN_BOOL ;
```

### Comments

Type used to designate Booleans used in the CAN API. A non-zero value indicates TRUE and zero indicates FALSE.

Top of list
Index page

# CAN_CONFIG

```
typedef struct CAN_CONFIG_
{
    unsigned int    Baud ;          // In kHz
    unsigned int    Mode ;          // Bit field, options
    unsigned int    Mask ;          // Bits to be affected
    unsigned int    Value ;         // Values to apply under mask.
    unsigned int    Bit_Rate_Div ;  // B10:B0 = (Clk prescaler - 1)
    // Bit_Rate_Div is ignored unless all the CAN_CFG_TSEG2_MASK
    //  and CAN_CFG_TSEG2_MASK bits are set in above Config_Mask word.
}
 CAN_CONFIG ;

#define CAN_PASSIVE_MODE        (0x2)
#define CAN_LOOP_BACK_MODE      (0x4)
#define CAN_TXQ1_DRIVES_ALL     (0x8)
#define CAN_TX_PRIORITY_QUEUES  (0x10)

#define CAN_CFG_OVERWRITE_LAST_MESSAGE (0x8000)
#define CAN_CFG_TSEG2_MASK             (0x7000)
#define CAN_CFG_TSEG1_MASK             (0x0F00)
#define CAN_CFG_AUTO_RESTART           (0x0010)
#define CAN_CFG_SYNC_JMP_WIDTH         (0x000C)
#define CAN_CFG_RX_SAMPLE_3_POINTS     (0x0002)
#define CAN_CFG_RX_EDGE_SYNC_MODE      (0x0001)
```

## Comments

The CAN_CONFIG structure is used in the **CAN_Open_Port()** and **CAN_Reconfig()** API's to specify a CAN ports bit timing configuration.

`Baud`

CAN baud rate, in kilo-Hertz. The standard CAN baud rates are (in kHz) 10, 20, 50, 100, 125, 250, 500, 800 and 1000. For these settings, the hardware can provide an exact fit baud rate. For other non-standard settings, there may be some slight error. The resulting error is observable through the **Baud**read out provide by the **CAN_Status()**API.

Note: The designator "Hz" is used here for "bits per second" (or *"bit cycles per second"*).

`Mode`

This bit field specifies which mode options to use. All bits set zero provide normal operation. Bits set indicate the respective option should be applied:

`CAN_PASSIVE_MODE`: Output is held at recessive level. The CAN module is only listening.

`CAN_LOOP_BACK_MODE`: An internal loopback between the two CAN channels is made. (This may be useful for software testing.)

`CAN_TXQ1_DRIVES_ALL`: This option specifies that the first CAN transmit FIFO designated with `CAN_TX1`should be fed into all hardware transmit registers (of which there are three total) that have a zero size software queue specified. For example, when this option is used with `Tx_Q_Size[CAN_TX2]`member of the **CAN_PORT_INIT**structure set to zero, then the `CAN_TX1`FIFO will feed the hardware transmit register that otherwise would have been associated with `CAN_TX2`. This option bit has no affect when used with the **CAN_Reconfig()**. The setting applied at the **CAN_Open_Port()**call remains in affect. Refer to the **CAN_Send()**API documentation for further explanation of this option.

CAN_TX_PRIORITY_QUEUES: This option sets up the CAN transmit API with two FIFO's. One queue is normal priority and a second queue is sent with high priority. No CAN messages will be loaded from the normal priority FIFO into the CAN hardware transmit registers unless the high priority FIFO is empty. The CAN messages from the high priority FIFO will be loaded first. The two queues are designated with the indexes CAN_TX_NORMALand CAN_TX_HIGH. These indexes replace the transmit channel designators CAN_TX1, CAN_TX2and CAN_TX3when operating in CAN_TX_PRIORITY_QUEUESmode. The Tx_Q_Size[CAN_TX_NORMAL]member of the **CAN_PORT_INIT**structure specifies the size of the normal priority queue and the Tx_Q_Size[CAN_TX_HIGH]member specifies the size of the high priority queue. (The Tx_Q_Size[CAN_TX3]member plays no role in this mode.) This option bit has no affect when used with the **CAN_Reconfig()**. The setting applied at the **CAN_Open_Port()**call remains in affect. Refer to the **CAN_Send()**API documentation for further explanation of this option. Note that the CAN_TXQ1_DRIVES_ALLoption bit is ignored when CAN_TX_PRIORITY_QUEUESis set.

Mask

This bit field specifies which option fields in Valueare applied. *Set this member to zero for default settings*(which is what most users are expected to do here). These optional settings are not expected to be needed by most users. They are only intended as provisional.

The CAN_CFG_TSEG1_MASKand CAN_CFG_TSEG2_MASKfields must both be controlled if either one is controlled. These in conjunction with the Bit_Rate_Divvalue are used to set non-standard baud rates and bit timing.

Value

This bit field specifies various options. Only fields with corresponding bits in Maskset to '1' are applied. All other fields are ignored.

CAN_CFG_OVERWRITE_LAST_MESSAGE: When channel's hardware receiver FIFO is full and a new message arrives, the newest message in the hardware FIFO (the previous message) is overwritten by the latest. In default mode, the latest message would be lost under these conditions.

CAN_CFG_TSEG2_MASK: This three bit field specifies the length (-1) of the second time segment. CFG_TSEG2=0 is not allowed. CFG_TSEG2=1 is only allowed in direct sampling mode.

CAN_CFG_TSEG1_MASK: This four bit field specifies the length (-1) of the first time segment. It includes the propagation time segment. CFG_TSEG1=0 and CFG_TSEG1=1 are not allowed.

CAN_CFG_AUTO_RESTART: When this bit is set to '1' the CAN is restarted automatically after 128 groups of 11 recessive bits, following a bus stop condition (This is the CAN driver software's default mode.) When this bit is zero, the CAN bus must be restarted manually. The **CAN_Reconfig()**API can be used to restart a bus.

CAN_CFG_SYNC_JMP_WIDTH: This two bit field specifies the synchronization jump width - 1. (By default this value is set to zero resulting in a synchronization jump width of 1 time quantum.)

CAN_CFG_RX_SAMPLE_3_POINTS: This bit set specifies that three sampling points per data bit with majority decision be used at the receiver. (This is the default.) With this bit cleared to zero, the receiver will use only one sampling point per data bit.

CAN_CFG_RX_EDGE_SYNC_MODE: This bit set specifies that both edges ("Recessive to Dominant" and vice-versa) are used by the receiver to synchronize to the incoming data. With this bit cleared to zero, only the "R to D" edge is used. (This is the driver's default.)

Bit_Rate_Div

This value will be applied directly to the CAN devices *Bit Rate Divisor Register*when both CAN_CFG_TSEG1_MASKand CAN_CFG_TSEG2_MASKbits in Maskare set. This is an 11 bit value in B10:B0. The B15:B11 upper bits should be zero. This advanced setting in conjunction with the CFG_TSEG1 and CFG_TSEG2 fields in the Valueword allow manual control over the CAN baud rate and bit sampling point. The resulting CAN baud rate can be calculated as follows:

baud = FREQUENCY / clk_scaler

where

FREQUENCY = CPU **processor frequency**
clk_scaler = tq_divisor * tq_per_bit
(Note: Notation "tq" means "Time Quanta" here.)
tq_divisor = (`Bit_Rate_Div`+ 1)
tq_per_bit = (1 + (CFG_TSEG1 + 1) + (CFG_TSEG2 + 1))

CFG_TSEG1 = 4 bit field `Value[B11:B8]`
CFG_TSEG2 = 3 bit field `Value[B14:B12]`

The bit sampling occurs (CFG_TSEG1 + 2) time quanta past the start of the bit period.

Note that the `Baud`member of this structure is ignored when the `CAN_CFG_TSEG1_MASK`and `CAN_CFG_TSEG2_MASK`in `Mask`indicate that baud should be set manually as specified here.

## Related Topics

API function **CAN_Open_Port()** - Open CAN port
API function **CAN_Reconfig()** - Reconfigure CAN port
API function **CAN_Send()** - Send CAN message

**Top of list**
**Index page**

---

# CAN_ERRORS

```
typedef struct CAN_ERRORS_
{
    unsigned char   TxErrCnt ;      // Bosch defined Tx Error Count
    unsigned char   RxErrCnt ;      // Bosch defined Rx Error Count

    unsigned int    BusStatus ;     // Bit field, see CAN_BS_* masks below.

} CAN_ERRORS ;

#define CAN_BS_RX96     (0x0008)        // Rx error counter >= 96
#define CAN_BS_TX96     (0x0004)        // Tx error counter >= 96
#define CAN_BS_BUS_OFF  (0x0002)        // Bus off flag
#define CAN_BS_PASSIVE  (0x0001)        // Error passive state (if not bus off)
```

## Comments

The `CAN_ERRORS` structure is used to report the bus error status of a specified CAN port. These fields are set by the **CAN_Error_Counts()** API.

`TxErrCnt`
This count is the transmitter error counter which operates according to the Bosch CAN specification. When the transmitter error count exceeds 255, this counter stops counting and remains latched at 255.

`RxErrCnt`
When not in bus off state, this counter holds the receiver error count which operates according to

the Bosch CAN specification. When in bus off state, this counter counts the idle states.

`BusStatus`
This bit field indicates the bus error conditions. The four lower bits are set to indicate error conditions as described by the CAN_BS_* mask defines.

`CAN_BS_RX96`: This bit is set if the receiver error counter is greater than or equal to 96.

`CAN_BS_TX96`: This bit is set if the transmitter error counter is greater than or equal to 96.

`CAN_BS_BUS_OFF`: This bit is set if the bus is in the bus off state.

`CAN_BS_PASSIVE`: The bus is in the "*error passive*" state if this bit is set and the `CAN_BS_BUS_OFF`bit is not set.

If both the `CAN_BS_BUS_OFF`and `CAN_BS_PASSIVE`bits are zero, then the bus is in its normal operation state ("*error active*").

**Top of list**
**Index page**

---

# CAN_ID

```
typedef struct CAN_ID_
{
    unsigned int    Normal ;
    unsigned int    Extended ;
} CAN_ID ;
```

## Comments

The `CAN_ID` structure is used in the API where ever CAN identifiers must be conveyed. This includes receiver filter initialization, message send and message receive.

`Normal`
The normal 11 bit CAN identifiers are placed here left justified in B15 through B5. When the extended 29 bit CAN identifiers are used, this field contains identifier bits ID28 through ID13.

`Extended`
When using the 29 bit extended CAN identifiers this field holds identifier bits ID12 through ID00 left justified in bits B15 through B3. **Note that for receiver filters' `Id_Mask`, this identifier bit field (B15:B3) must be set to all 1's in order to receive normal 11 bit CAN identifiers**(=0xFFF8, or =0xFFFA if the `CAN_FILT_RTR`flag mentioned below is OR'ed in).

When used to initialize receiver filters, this word also carries the following input message qualifiers in two of the lower three bits:

`CAN_FILT_IDE`= 0x4: Accept only 29 bit extended ID messages through this filter.
`CAN_FILT_RTR`= 0x2: Accept only Remote Transmit Request (RTR) messages through this filter.

The lower three bits of this word are void for send/receive message identifiers.

## Related Topics

API function **CAN_Rx_Filters()** - Set receiver filters
API function **CAN_Send()** - Send CAN message
API function **CAN_Recv()** - Get received message

**Top of list**
**Index page**

---

# CAN_MSG

```
typedef struct CAN_MSG_
{
    CAN_ID          Id ;        // 11 or 29 bit identifier
    unsigned char   Data[8] ;   // 0..8 bytes data
    unsigned int    Len_Ctrl ;  // Data Length Count (and some flags)

} CAN_MSG ;

  // Masks for CAN_MSG.Len_Ctrl word
#define CAN_DLC_FIELD  (0x000F) // Data Length Count, [0..8]
#define CAN_IDE_FLAG   (0x0010) // Extended (29 bit) ID flag
#define CAN_RTR_FLAG   (0x0020) // Remote Request flag
#define CAN_RX_AF1     (0x0100) // Received through acceptance filter 1
#define CAN_RX_AF2     (0x0200) // Received through acceptance filter 2
#define CAN_RX_AF3     (0x0400) // Received through acceptance filter 3
```

## Comments

The `CAN_MSG` structure provides a container for outgoing and incoming CAN messages.

`Id`
: This **structure**carries the message's identier.

`Data`
: The CAN message data bytes are put here, when message contains data.

`Len_Ctrl`
: The primary purpose of this member is to state the "Data Length Count", which is the number of data bytes in the CAN message. The lower four bits specify this count. The legal range is from 0 to 8 bytes, inclusive. (Illegal values here on transmit messages will be output verbatim, but no more than 8 data bytes will be transmitted.)

  Two other discrete flags here indicate if the message uses a 29 bit identifier (`CAN_IDE_FLAG`) or is a "Remote Transmit Request" trigger (`CAN_RTR_FLAG`) when set.

  For received messages, the three `CAN_RX_AFx`discrete flags indicate which of the acceptance filters permitted this message to come in.

## Related Topics

API function **CAN_Send()** - Send CAN message
API function **CAN_Recv()** - Get received message
API function **CAN_Peek()** - Peek at received message

**Top of list**

# CAN_PORT_INIT

```
typedef struct CAN_PORT_INIT_
{
    CAN_BOOL        fDisable_Rx ;   // Initially disable Recvr
    CAN_CONFIG      Config ;        // Configuration Reg settings
    unsigned int    Rx_Q_Size ;     // Number of CAN msgs
    unsigned int    Tx_Q_Size[NUM_CAN_TX] ;
} CAN_PORT_INIT ;
```

## Comments

`fDisable_Rx`
Set this flag if you do not want the receiver immediately enabled on return from the **CAN_Open_Port()**call.

`Config`
This data **structure**specifies the baud rate and other parameters.

`Rx_Q_Size`
This count specifies the size of the receiver ring buffer which will be created for this CAN channel. Legal values are in the range from 1 to 500 (inclusive) messages.

`Tx_Q_Size`
This array of three counts provides individual control over the lengths of the transmit ring buffers which feed the three hardware transmit registers. Legal values are in the range from 0 to 500 (inclusive) messages. When operating with a transmit ring buffer size zero, the **CAN_Send()**API will either load the hardware transmit register directly or return without taking any action. If `CAN_TX2`and `CAN_TX3`FIFO's are specified with zero size when the `CAN_TXQ1_DRIVES_ALL`**option**flag is set in the `Config.Mode`word, then the hardware transmit register (TX2, TX3) normally driven by the FIFO corresponding with that `Tx_Q_Size`value will instead be loaded from the `CAN_TX1`FIFO. In this manner, a single output queue can be used to feed more than one hardware transmit register.

When the `CAN_TX_PRIORITY_QUEUES`**option**in the `Config.Mode`word flag is used, then the alternate indexes `CAN_TX_NORMAL`and `CAN_TX_HIGH`apply here.

## Related Topics

API function **CAN_Open_Port()** - Open CAN port
API function **CAN_Send()** - Send CAN message

Top of list
Index page

# CAN_RX_FILT

```
typedef struct CAN_RX_FILT_
{
    // Mask 1 bits = Don't care
    // Mask 0 bits = Must match specified values
    CAN_ID          Id_Mask ;       //
    unsigned int    Data_Mask ;     // 2 byte data mask
    CAN_ID          Id_Value ;      //
    unsigned int    Data_Value ;    // 2 byte data mask

} CAN_RX_FILT ;
```

## Comments

The `CAN_RX_FILT` structure is used to initialize one of the CAN device's three receiver filters. By setting a receiver filter's mask registers to all bits '1' (="Don't care"), the filter will accept all incoming CAN messages.

`Id_Mask`
: This **structure**specifies which bits of the received messages identifier should be compared. Bits set to '0' indicate bits that should be compared against those in `Id_Value`. Note that to receive normal 11 bit CAN identifiers, the 13 bit field B15:B3 in the `Extended`member of this structure must be set to all 1's (=0xFFF8).

`Data_Mask`
: Incoming messages can also be filtered based on contents of first two of the up to 8 bytes of CAN message data. This 16 bit field is used to mark which data bits of incoming messages should be compared against the corresponding bits in `Data_Value`. The lower byte corresponds to first CAN data byte and the upper byte corresponds to the second data byte. Zero bits are compared and bits set to '1' are don't care.

`Id_Value`
: This **structure**specifies the identifier bit values that will be compared against those received. Only the bits set to zero in the `Id_Mask`are applied in the filter test.

`Data_Value`
: These 16 bits are compared against the first two CAN data bytes. Only the bits set to zero in the `Data_Mask`are applied in the filter test.

## Related Topics

API function **CAN_Rx_Filters()** - Set receiver filters

**Top of list**
**Index page**

---

# CAN_STATUS

```
typedef struct CAN_STATUS_
{
    CAN_BOOL        fPort_Opened ;  // TRUE if port is now open
    unsigned long   Baud ;          // In bits/second
    unsigned int    Device_Enable ; // Bit field for Rx & 3 Tx.
    unsigned int    Errors ;        // Bit field
    unsigned int    Rx_Count ;      // Messages in ring buffer
    unsigned int    Tx_Count[NUM_CAN_TX] ;

} CAN_STATUS ;

#define CAN_RX_SEL   (0x8000)   // Receiver control flag
#define CAN_TX1_SEL  (0x0800)   // TX1 reg control flag
#define CAN_TX2_SEL  (0x1000)   // TX2 reg control flag
#define CAN_TX3_SEL  (0x2000)   // TX3 reg control flag

#define CAN_ARBLOSS     0x0004      // Arbitration lost during tx
#define CAN_OVRLOAD     0x0008      // Overload condition
#define CAN_RX_OVR_ERR  0x0010      // Receiver overrun in hardware
#define CAN_BIT_ERR     0x0020      // Bit error during tx or rx
#define CAN_STUF_ERR    0x0040      // Stuffing error during tx or rx
#define CAN_ACK_ERR     0x0080      // Ack error during tx or rx
#define CAN_FORM_ERR    0x0100      // Format error during tx or rx
#define CAN_CRC_ERR     0x0200      // CRC error during tx or rx
#define CAN_BUS_OFF     0x0400      // CAN switched to bus off state
#define CAN_SW_RX_OVR   0x0800      // Software Rx ring buffer overflow.
```

## Comments

The CAN_STATUS structure is used to report the status of a specified CAN port. These fields are set by the **CAN_Status()** API.

`fPort_Opened`
This flag is set to one (TRUE) if the respective port is opened, otherwise it is set to zero (FALSE).

`Baud`
The actual baud rate (assuming an accurate clock crystal) is reported here. This may differ slightly from the **requested**baud rate when a best fit must be used (for non-standard CAN baud rates) instead of an exact fit. Note that this value is resolved to Hz (not kHz as was the requested value).

Note: The designator "Hz" is used here for "bits per second" (or *"bit cycles per second"*).

`Device_Enable`
This bit field reports which devices are enabled. The following bits are set when the corresponding device is enabled:

CAN_RX_SEL= 0x8000 - Receiver enabled
CAN_TX1_SEL= 0x0800 - Transmit register #1 enabled
CAN_TX2_SEL= 0x1000 - Transmit register #2 enabled
CAN_TX3_SEL= 0x2000 - Transmit register #3 enabled

`Errors`
This bit field indicates the errors which have occurred. The bit definitions are listed above with the defined constants (e.g. CAN_ARBLOSS). The internal error flags are cleared upon this read out.

`Rx_Count`
This count indicates the number of received CAN messages available for **pickup**.

`Tx_Count`
These three counts indicate the number of CAN messages awaiting output on each of the three CAN transmit registers. This count includes any CAN message pending output in a hardware transmit register plus those awaiting in the driver's software transmit FIFO.

## Related Topics

API function **CAN_Status()** - Check status

**Top of list**
**Index page**

---

End of document

---




# @CHIP-RTOS - CAN Error Codes



## CAN Error Codes

These error codes are returned by the CAN **API** in the **AX** register.

`CAN_EC_SUCCESS` = 0, no problem
`CAN_EC_INVALID_PARAMS` = -1, Input parameters invalid
`CAN_EC_INVALID_BAUD` = -2, Invalid baud rate specified
`CAN_EC_PORT_NOT_OPENED` = -3, Op not allowed on closed port
`CAN_EC_PORT_ALREADY_OPENED` = -4, Attempt to open port already opened
`CAN_EC_NO_BUF_SPACE` = -5, Transmit buffer is full
`CAN_EC_TIMEOUT` = -6, Event wait time-out
`CAN_EC_LOW_RESOURCES` = -7, RTX event group allocation failed
`CAN_EC_NO_SUCH_SERVICE` = -8, Invalid index in AH register
`CAN_EC_INVALID_Q_SIZE` = -9, Queue **size** out of range
`CAN_EC_LOW_MEMORY` = -10, Heap memory allocation failed

End of document

## CAN_Send

Queue a CAN message for transmission.

```
int CAN_Send (  unsigned int port_idx,
                unsigned int channel,
                const CAN_MSG far *msg );
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*channel*

Transmit register specifier:
`CAN_TX1`= 0
`CAN_TX2`= 1
`CAN_TX3`= 2

or when operating in **CAN_TX_PRIORITY_QUEUES** mode:
`CAN_TX_NORMAL`= 0
`CAN_TX_HIGH`= 1
(2 works same as CAN_TX_HIGH here)

*msg*

Pointer to a **CAN_MSG** data structure holding contents of CAN message to be transmitted, set by caller.

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success, message has been transferred into specified transmit queue.
`CAN_EC_INVALID_PARAMS` = -1: `port_idx`, `filter_idx` invalid or NULL `msg` pointer.
`CAN_EC_PORT_NOT_OPENED` = -3: Port has not been **opened**. `CAN_EC_NO_BUF_SPACE` = -5: Transmit queue for this output channel is full, message not accepted.

## Comments

Three different modes of operation are available for the CAN transmission, selected at the **CAN_Open_Port** API call:

1. **Default mode**

This mode uses three transmit software FIFO queues, where each FIFO feeds a specific hardware transmit register. The size of each FIFO is configurable.

2. **CAN_TX_PRIORITY_QUEUES Mode**

   This mode is selected by setting the CAN_TX_PRIORITY_QUEUES bit in the **CAN_CONFIG** structure's Mode word. In this mode two transmit FIFO's are used, one with normal priority and one with high priority. The size of the normal priority queue is specified by the Tx_Q_Size[CAN_TX_NORMAL] value in the **CAN_PORT_INIT** structure at the **CAN_Open_Port** call. The high priority queue size is specified here with the Tx_Q_Size[CAN_TX_HIGH] value.

3. **CAN_TXQ1_DRIVES_ALL Mode**

   This mode is selected by setting the CAN_TXQ1_DRIVES_ALL bit in the **CAN_CONFIG** structure's Mode word. In this mode, a single queue feeds more than one of the hardware transmit registers. If a queue size of zero is specified for channels CAN_TX2 and CAN_TX3, then the respective hardware transmit registers will be fed from the common FIFO. If both the Tx_Q_Size[CAN_TX2] and Tx_Q_Size[CAN_TX3] values in the **CAN_PORT_INIT** structure are zero when **CAN_Open_Port** is called, then the single FIFO whose size was specified by Tx_Q_Size[CAN_TX1] drives all three hardware transmit channels.

**Comments General to All Operation Modes**

The CAN_Send( ) function either places the provided CAN message directly into the hardware register when there is no message currently pending output already in this register, or into the software ring buffer queue. When no more space is available in the output queue, an error code CAN_EC_NO_BUF_SPACE is returned and no operation is performed.

The driver feeds the output queues into the hardware transmit registers in an interrupt driven fashion. The hardware prioritizes the three transmit registers based on "first recessive bit" logic, similar to that used for normal CAN bus arbitration. However in this case, the "arbitration" is performed internally prior to presentation of the outgoing CAN message on the bus. This priorization mechanism can lead to transmission which is not done in FIFO order for the operational modes which drive more than one hardware transmit register from a single FIFO.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

**Comments on Default Mode**

The CAN driver provides three queues for outgoing CAN messages. Each queue is fed into one of the three hardware transmit registers provided by the CAN device.

The output queues can be loaded with this API prior to enabling the respective hardware transmit register. In this case no output from this channel will occur until the channel is enabled with the appropriate channel enable command to the **CAN_Control()** function.

**Comments on CAN_TX_PRIORITY_QUEUES Mode**

The CAN_TX_PRIORITY_QUEUES option bit in the **CAN_CONFIG** Mode word selects this mode. CAN messages from the CAN_TX_NORMAL queue will only be loaded into the CAN hardware transmit registers when the CAN_TX_HIGH queue is empty. The CAN messages in the CAN_TX_HIGH queue are higher priority and will be loaded before those in the CAN_TX_NORMAL queue.

When using this CAN_Send( ) API, the channel parameter is used to specify either the normal (CAN_TX_NORMAL) or high (CAN_TX_HIGH) priority queues.

Note that when this API is called with channel set to CAN_TX_HIGH, any CAN messages from the CAN_TX_NORMAL queue which have already been loaded into the hardware registers will remain there. Consequently, it can happen that pending CAN messages from the CAN_TX_NORMAL queue that have already been loaded into one of the three hardware transmit registers may be transmitted after calling this API with a CAN_TX_HIGH message and before this higher priority message is loaded into the transmit

registers by the CAN driver. The new high priority message will only be loaded when one of the three transmit registers becomes available (empty).

Once loaded into the hardware register, the CAN message recessive bit prioritization logic is used by hardware to select among the three transmit registers. This too may lead to messages from the normal priority queue being transmitted ahead of a high priority CAN message, if the high priority queue's message does not have a high priority CAN message ID (low number). For applications where this behavior is not tolerable, you might consider disabling all but one of the transmit registers. However, this may lead to performance problems so should be avoided when possible. The queues will drain through what ever transmit registers have been enabled.

**Comments on CAN_TXQ1_DRIVES_ALL Mode**

This mode is selected by setting the CAN_TXQ1_DRIVES_ALL option bit in the **CAN_CONFIG** Mode word. (The CAN_TX_PRIORITY_QUEUES bit must be zero.) In this mode the hardware transmit registers CAN_TX2 and/or CAN_TX3 can be driven from the CAN_TX1 FIFO by specifying zero **sizes** for the CAN_TX2 and/or CAN_TX3 software FIFO's.

Example #1:

```
CAN_PORT_INIT init ;
init.Config.Mode = CAN_TXQ1_DRIVES_ALL ;
init.Tx_Q_Size[CAN_TX1] = 20 ;
init.Tx_Q_Size[CAN_TX2] = 5 ;
init.Tx_Q_Size[CAN_TX3] = 0 ;
```

These settings would cause TX1 and TX3 hardware transmit registers to both be driven from the 20 message CAN_TX1 FIFO. The TX2 hardware transmit register is connected to its private 5 message queue.

Example #2:

```
CAN_PORT_INIT init ;
init.Config.Mode = CAN_TXQ1_DRIVES_ALL ;
init.Tx_Q_Size[CAN_TX1] = 20 ;
init.Tx_Q_Size[CAN_TX2] = 0 ;
init.Tx_Q_Size[CAN_TX3] = 0 ;
```

These settings would cause all three hardware transmit registers to be filled from the 20 message CAN_TX1 FIFO.

When the CAN_TXQ1_DRIVES_ALL option is used to drive more than one hardware transmit register from the first software transmit FIFO, the CAN transmissions may not necessarily come out in FIFO order. This is due to the CAN bus arbitration based on CAN label performed within the CAN port device's set of transmit registers. A typical scenerio would be that the CAN driver software takes three CAN messages from the send FIFO and places these into the three hardware transmit registers. Then the hardware transmit arbitration logic selects the CAN message among these three with the lowest CAN ID for transmission.

When the CAN_TXQ1_DRIVES_ALL option is used to drive CAN_TX2 and CAN_TX3 transmit registers, calling this CAN_Send( ) API with the channel argument set to either CAN_TX2 or CAN_TX3 performs nearly the same operation as CAN_TX1. In the case where the provided CAN message is placed in the single send FIFO, the functionality is identical. When the FIFO and specified transmit register are empty, there is a slight difference. In this case the channel argument specifies which hardware transmit registers is loaded.

## See Also

**CAN_Recv** **CAN_Control** **CAN_Open_Port**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Recv

Receive a CAN message, with optional time-out.

```
int CAN_Recv (  unsigned int port_idx,
                    CAN_MSG far *msg,
                    int timeout );
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*msg*

Output Parameter: Pointer to a **CAN_MSG** data structure were this API will output the next CAN message if one is available.

*timeout*

Time-out
Negative value: No wait, return immediately
Zero: Wait forever for a CAN message
Positive value: Maximum milliseconds to wait for a CAN message

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success, new message is in data structure referenced by `msg` pointer.
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` or `msg` pointer invalid.
`CAN_EC_PORT_NOT_OPENED` = -3: Port has not been **opened**. `CAN_EC_TIMEOUT` = -6: Time period expired and still no message available.

## Comments

This function is used to pick up the next received CAN message from the receiver FIFO queue. The message is removed from the input queue upon transfer here to the user.

The `timeout` parameter allows the caller to optionally sleep on the receiver a specified number of milliseconds until a received CAN message becomes available.

Note that if the CAN port is closed, the **CAN_Close_Port** API strobes the event bits to assure that a program thread which is sleeping on the receiver within this API will return (and not be left hanging on a no longer existant event group).

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

**Important Note:**

No CAN messages will be received until the port's receiver filters have been enabled for reception using the **CAN_Rx_Filters**API. Following a system reset, the CAN receiver filters are initialized to reject all incoming CAN messages by default.

## See Also

**CAN_Peek** **CAN_Send** **CAN_Open_Port**
**CAN_Close_Port** **CAN_Rx_Filters**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Peek

Look at next CAN message, but leave it in queue.

```
int CAN_Peek (  unsigned int port_idx,
                    CAN_MSG far *msg );
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*msg*

Output Parameter: Pointer to a **CAN_MSG** data structure were this API will copy the next CAN message from receiver queue, if one is available.

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success, a copy of next CAN message in queue is in the data structure referenced by the `msg` pointer.
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` or `msg` pointer invalid.
`CAN_EC_PORT_NOT_OPENED` = -3: Port has not been **opened**. `CAN_EC_TIMEOUT` = -6: No message available, receiver queue is empty.

## Comments

This function picks up a copy of the next received CAN message from the receiver queue FIFO without removing that message from the queue. This function always returns immediately. If no message was available, the return value is `CAN_EC_TIMEOUT`.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

**Important Note:**

No CAN messages will be received until the port's receiver filters have been enabled for reception using the **CAN_Rx_Filters**API. Following a system reset, the CAN receiver filters are initialized to reject all incoming CAN messages by default.

## See Also

CAN_Recv CAN_Rx_Filters CAN_Open_Port

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Reconfig

Adjust baud rate or bit timing parameters on an open CAN port.

```
int CAN_Reconfig ( unsigned int port_idx,
                   const CAN_CONFIG far *config);
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*config*

Pointer to a **CAN_CONFIG** data structure containing requested bit timing (baud) and control information set by caller.

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success, new settings were accepted.
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` or `config` pointer is invalid.
`CAN_EC_INVALID_BAUD` = -2: Invalid baud rate or bit timing specification.
`CAN_EC_PORT_NOT_OPENED` = -3: Port has not been **opened**.

## Comments

This function allows bit timing to be adjusted on a CAN port which has already been opened. (Under normal circumstances it is not expected that this function will be used.)

This function may also be used to manually switch back on a CAN bus following a **bus off** event. By default, this is performed automatically by the driver. However the user may specify a manual **restart** mode.

A successful call to this API zeroes the port's `CANn_EV_BUS_OFF` event.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## See Also

**CAN_Open_Port**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Status

Read out status information for specified CAN port.

```
int CAN_Status ( unsigned int port_idx,
                 CAN_STATUS far *sts );
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*sts*

Pointer to a **CAN_STATUS** data structure that will be written to by this API to report the status information for the specified CAN port.

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` invalid or `sts` pointer invalid

## Comments

This function reports various status information concerning the specified CAN port. After reading out the reported error flags, the internal error flag registers are cleared (a destructive read).

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## See Also

**CAN_Control** **CAN_Open_Port** **CAN_Close_Port**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Control

Enable or disable specified channels. Purge specified FIFO's.

```
int CAN_Control ( unsigned int port_idx,
                  unsigned int enables,
                  unsigned int disables,
                  unsigned int purge_fifos ) ;
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*enables*

Bit field indicating which devices to enable.

*disables*

Bit field indicating which devices to disable.

*purge_fifos*

Bit field indicating which FIFO's to clear.

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` is invalid
`CAN_EC_PORT_NOT_OPENED` = -3: Port has not been **opened**.

## Comments

This function provides a way to enable or disable the receiver and the three individual transmit register channels. Also, any or all of the receiver or transmit FIFO's can be cleared.

The **CAN_Open_Port()** enables all three transmitters and optionally the receiver as well. So this provisional API here is not necessarily required for CAN port operation.

The input parameters `enables`, `disables` and `purge_fifos` are each bit fields with the identical bit mapping as follows:

CAN_RX_SEL= 0x8000, Receiver control flag
CAN_TX1_SEL= 0x0800, TX1 register control flag
CAN_TX2_SEL= 0x1000, TX2 register control flag
CAN_TX3_SEL= 0x2000, TX3 register control flag

A bit set indicates that the action associated with the particular CPU register should be taken for the device designed by the bit. For example:

```
CAN_Control(CAN0_PORT,
            CAN_TX3_SEL,         // = enables
            CAN_TX2_SEL,         // = disables
    CAN_RX_SEL | CAN_TX2_SEL) ; // = purge_fifos
```

This example API call will enable transmit register #3. It will disable transmit register #2. It will purge (delete the contents of) the receiver FIFO and the FIFO associated with transmit register #2.

When transmit FIFO's are purged, any CAN message pending output in the specified hardware transmit register is also aborted.

In the non-sense case where a device's bit is set in both the `enables` and `disables` input parameters, indicating that both an enabling and disabling action should be taken on the same device, the enabling will take precedence and the device will be enabled.

Note that when the `CAN_TX1` transmit FIFO is feeding more than just the TX1 transmit register (`CAN_TXQ1_DRIVES_ALL` **configuration** option), all of the transmit registers fed must be blocked with the `disables` mask in order to stop transmission from this FIFO.

e.g.

```
disables = CAN_TX1_SEL | CAN_TX2_SEL | CAN_TX3_SEL ;
```

would be needed to stop transmission from the `CAN_TX1` FIFO if all three transmit registers were being fed from this FIFO. And likewise, bits for all channels associated with the queue would need to be set in the `enables` to obtain a full bandwidth draining of the queue into the available hardware transmit registers.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## See Also

**CAN_Open_Port** **CAN_Status**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Close_Port

Close down a specified CAN port.

```
int CAN_Close_Port ( unsigned int port_idx );
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` invalid

## Comments

This function closes a CAN port, releasing any resources it was using (e.g. heap memory for FIFO's and RTX event group). Prior to releasing the **event group**, the event bits in the RTX event group associated with this port are **strobed** in order to release any program threads which were **waiting** on events for this port which is now being closed.

Any user **callback** vectors that had been set for this port will be removed.

The CAN device is deactivated. Any pending transmission is aborted. Aborting a pending transmission requires some additional time beyond when this API returns such that if the port were to be immediately reopened, a subsequent call to **CAN_Send** could fail due to a transmit buffer still busy condition.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## See Also

**CAN_Open_Port** **CAN_Event_Sleep** **CAN_Recv**
**CAN_Set_Callback**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Event_Sleep

Await any one event from a specified set of CAN port events.

```
int CAN_Event_Sleep ( unsigned int events,
                      int timeout );
```

## Parameters

*events*

Bit field specifying a set of events to await

*timeout*

Time-out
Negative value: No wait, return immediately
Zero: Wait forever for the CAN event(s)
Positive value: Maximum milliseconds to wait for the CAN event(s)

## Return Value

= 0 indicates no events occurred (time-out)
non-zero = event bits which came true among those that were specified in `events` input parameter

## Comments

This function sleeps on the event group used internally by the CAN driver, awaiting specified CAN events. Note that the events can be on either of the two CAN ports. The events specified in the bit field input parameter `events` can be any combination of the following:

CAN0_EV_RX_RDY = 0x0001, CAN0 Receiver data ready
CAN0_EV_TX1_EMP = 0x0002, CAN0 Transmitter #1 empty
CAN0_EV_TX2_EMP = 0x0004, CAN0 Transmitter #2 empty
CAN0_EV_TX3_EMP = 0x0008, CAN0 Transmitter #3 empty
CAN0_EV_BUS_OFF = 0x0010, CAN0 Bus off condition

CAN1_EV_RX_RDY = 0x0100, CAN1 Receiver data ready
CAN1_EV_TX1_EMP = 0x0200, CAN1 Transmitter #1 empty
CAN1_EV_TX2_EMP = 0x0400, CAN1 Transmitter #2 empty
CAN1_EV_TX3_EMP = 0x0800, CAN1 Transmitter #3 empty
CAN1_EV_BUS_OFF = 0x1000, CAN1 Bus off condition

This function returns when any one of the specified events occurs. If no events are specified the function returns immediately with a return value zero. Any event bits in input parameter `events` which apply to a port which is not open will be ignored and treated as zeros.

This function has no affect on the event flags. Event flags are ***not*** cleared as a side affect of this function call.

The `CANn_EV_RX_RDY` event is cleared after the respective receiver queue is emptied by calls to the **CAN_Recv()** API. The `CANn_EV_BUS_OFF` event is cleared by calls to **CAN_Open_Port()**, **CAN_Reconfig()** or **CAN_Close_Port()** API.

The transmit event flags operate differently depending on whether or not the `CAN_TXQ1_DRIVES_ALL` or `CAN_TX_PRIORITY_QUEUES` modes are being used. When either of these transmit modes are used, the **CAN_Send** API clears the transmit flags for all channels associated with the software queue that the output CAN message is directed to. And this same group of transmit channel event flags will be set to '1' simulataneously when the software queue(s) feeding those channels is (are) emptied. However there may still be a CAN message or two awaiting output in the CAN hardware transmit registers after the events are set.

When the transmitters are operated in neither `CAN_TXQ1_DRIVES_ALL` nor `CAN_TX_PRIORITY_QUEUES` mode, the **CAN_Send** API clears the single event flag bit associated with the channel that the CAN message is directed. Then the respective transmit event bit will be set to '1' after the hardware register is emptied (CAN message sent) and no further queued messages were pending for this transmit device.

A port closing will cause this function to return if any event wait bits applied to that port which closed. In this case the event flags returned for the closed port should be ignored, since they were created artificially.

The handle to the event group which the CAN driver uses to implement this API can be retrieved with the **RTX_Find_EventGroup()** API provided that the user has not stepped on the name given this group with yet another event group by the same name. The name given this event group by the CAN driver is "_CAN". This event group exists only so long as one of the two CAN ports is **opened**.

An event group provides 16 independent events, of which this CAN driver uses ten (five events for each of two CAN ports). These ten event bits are specified above. The other six event flags are available to the user. If the user chooses to use these additional event flags for their own purposes then they should call directly the **RTX_Wait_For_Event()** rather then using this API. When this `CAN_Event_Sleep()` API is used, any event flags outside the scope of the ten used by the CAN driver are ignored. Furthermore, this API only waits for event flag '1' conditions. Using the `RTX_Wait_For_Event()` API directly, the user would have more flexibility.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## See Also

**CAN_Open_Port** **CAN_Recv** **CAN_Set_Callback**
**CAN_Close_Port** **CAN_Send** **CAN_Reconfig**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_install_rtx_isr

Install RTX Interrupt Service Routine

```
InterruptHandler hal_install_rtx_isr ( unsigned short irq,
                                       unsigned short count,
                                  InterruptHandler handler );
```

## Parameters

*count*

Number of interrupts generated before new user interrupt service routine is called.
count = 0 disables the user ISR (same as a NULL in handler)

*handler*

far pointer to user interrupt service routine
if pointer is NULL user ISR is disabled

## SC1x Parameters

*irq*

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) / DMA Interrupt Channel 0 (if DMA is used)
6 = INT6 (external) / DMA Interrupt 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 = *(*) DMA Interrupt Channel 2 (internal, not on SC12)*
14 = *(*) DMA Interrupt Channel 3 (internal, not on SC12)*
15 = NMI Powerfail Interrupt (internal/external)

(*) = used internally by @CHIP-RTOS, not available for user interrupt service functions

## SC1x3 Parameters

*irq*

HAL interrupt number from following list:

0 = *(*) Network controller 0 (internal)*
1 = INT1 (external)
2 = SPI controller (internal)
3 = INT3 (external) / *(*) USB controller (internal)*
4 = *(*) Serial port 2 (internal)*
5 = DMA Interrupt Channel 0
6 = DMA Interrupt Channel 1
7 = INT5 (external) / *(*) Serial port 3 (internal)*
8 = Timer0 (internal)
9 = Timer1 (internal)
10 = *(*) Timer 1ms (internal)*
11 = *(*) Serial port 0 (internal)*
12 = *(*) Serial port 1 (internal)*
13 = DMA Interrupt Channel 2
14 = DMA Interrupt Channel 3
15 = PFI Powerfail Interrupt (internal/external)
16 = *(*) CAN0 controller (internal) / (*) CAN1 controller (internal)*
17 = I2C controller (internal)
18 = Network controller 1 (internal)

(*) = used internally by @CHIP-RTOS, not available for user interrupt service functions

## SC23 Parameters

*irq*

HAL interrupt number from following list:

0 = *(*) Network controller 0 (internal)*
1 = INT1 (external)
2 = SPI controller (internal)
3 = INT3 (external) / *(*) USB controller (internal)*
4 = *(*) Serial port 2 (internal)*
5 = *(*) DMA Interrupt Channel 0*
6 = *(*) DMA Interrupt Channel 1*
7 = INT5 (external) / *(*) Serial port 3 (internal)*
8 = Timer0 (internal)
9 = Timer1 (internal)
10 = *(*) Timer 1ms (internal)*
11 = *(*) Serial port 0 (internal)*
12 = *(*) Serial port 1 (internal)*
13 = *(*) DMA Interrupt Channel 2*
14 = *(*) DMA Interrupt Channel 3*
15 = PFI Powerfail Interrupt (internal/external)
16 = *(*) CAN0 controller (internal) / (*) CAN1 controller (internal)*
17 = I2C controller (internal)
18 = *(*) Network controller 1 (internal)*

(*) = used internally by @CHIP-RTOS, not available for user interrupt service functions

## Return Value

Far pointer to old ISR handler

## 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 can be declared in either of the following forms:

```
void huge My_ISR(void) ;     // More efficient form
```

```
void interrupt My_ISR(void) ;     // Tolerated form
```

The more efficient huge procedures are recommended. The user ISR function must preserve only the DS and BP registers, so there is no requirement for the full register save/restore provided by the `interrupt` declarations.

Any required EOI signal is issued by the system ISR which calls your user ISR function. This EOI is issued after your function returns.

RTX ISR: If you install a RTX ISR you can use the follwing RTX calls in your ISR:

**RTX_Restart_Task**
**RTX_Wakeup**
**RTX_Signal_Sem**
**RTX_Signal_Events**
**RTX_Send_Msg**
**RTX_Start_Timer**
**RTX_Stop_Timer**
**RTX_Remove_Timer**

**Important Notes:**

The RTX ISR is slower than a normal ISR.

The `InterruptHandler` vector type is defined as:

```
typedef void huge (far *InterruptHandler)(void);
```

## SC1x Comments

Using RTX ISR are not recommended for INT5 or INT6 if DMA is used by the **Fossil** serial ports interface, because the slower RTX ISR could result in UART receiver character loss.

If you are using a RTX ISR for timer0, timer1, INT5 or INT6 there must not exist a normal ISR on the system which enables the interrupts during its execution!

Do not install a RTX ISR for the powerfail interrupt!

**Also important** : The NMI function of the multifunction pin 17 (RESET/NMI/LINK_LED) of the IPC@CHIP® SC11/SC12/SC13 is for power fail purposes only. It is not possible to use NMI as a "normal" interrupt pin like INT0 for generating interrupts. It can only be used as described in the IPC@CHIP® hardware documentation.

## SC1x3 Comments

INT1 and network controller 1 share the same interrupt. Due to this fact, the installed user interrupt service routines for INT1 and network controller 1 must be of same type. If the network controller 1 is used both ISRs must be of type RTX ISR.

## See Also

**hal_install_isr** **pfe_enable_int** **RTX_Install_RTI_Hook**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Set_Callback

Install a callback function to be called on specified event(s) within CAN interrupt service routine (ISR).

```
int CAN_Set_Callback ( unsigned int     port_idx,
                       CAN_ISR_CALLBACK vector,
                       unsigned int     events) ;
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*vector*

User procedure address, or NULL to remove a callback previously installed.

*events*

Bit field specifying on which CAN events the callback should be made.

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` invalid

## Comments

This function either installs or removes a user callback procedure that is called from within the CAN driver's Interrupt Service Routine (ISR) upon specified events. The `vector` argument can be set to NULL to remove existing vectors. Note that the **CAN_Close_Port** will remove any callbacks established for the CAN port being closed.

The callback will be made from within the CAN driver's interrupt service routine, which has been **registered** with the RTX @Chip-RTOS.

The system maintains three independent callbacks for each CAN port:

1. Receiver event callback: `CAN_CALLBACK_RX`
2. Transmit event callback: `CAN_CALLBACK_TX`
3. Bus off error event callback: `CAN_CALLBACK_ERR`

The `events` argument determines which callbacks are set by this API. The masks defined for this

purpose in CAN_API.H are indicated above. Combinations of these masks can be formed by OR'ing them together when it is desired that events share the same user callback. For example, the `CAN_CALLBACK_ALL` mask will set all three callbacks to a single provided vector.

The set of three callbacks was implemented so that less logic is required inside the user callback to figure out why it had been called, when independent callback functions are used. Note that all three transmit events are mapped to a single transmit callback.

The prototype for the user callback is as follows.

```
void huge My_Callback(unsigned int event,
                      const CAN_MSG far *msg) ;
```

The `event` parameter in this callback will be a single bit that indicates the reason for the callback. This bit is the same as used for the **CAN_Event_Sleep()** API's event masks, `CAN0_EV_RX_RDY` through `CAN1_EV_BUS_OFF`. Note that which CAN channel the event occurred on is conveyed by this event bit.

The `msg` pointer argument passed to the bus error callback is always NULL. No CAN message is associated with this callback. For receive events, the `msg` pointer references the CAN message that was just received. For transmit events, the `msg` pointer references the CAN message that just completed transmission.

The user callback function must preserve the DS, SS and SP registers. All other registers the callback is free to modify. The code within this callback must be keep to a minimum. This callback is performed within the CAN interrupt service. Interrupts are masked on entry to the callback and must remain masked. To encourage assembly language programming of this time critical callback, the parameters are also made available in registers:

- CX - Contains the `event` parameter.
- ES:DI - Contains the `msg` far pointer input parameter

These parameters are also pushed onto the stack to support a standard C language interface.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

**Note:**

Use of callbacks may overload the system when CPU is operated in **power save mode**and CAN is operated at high baud rates.

## See Also

**CAN_Close_Port** **CAN_Event_Sleep** **CAN_Recv**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.06 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.05 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CAN_Error_Counts

Read out CAN Rx/Tx error count and errror status information for specified CAN port.

```
int CAN_Error_Counts ( unsigned int port_idx,
                       CAN_ERRORS far *errors );
```

## Parameters

*port_idx*

CAN port specifier: `CAN0_PORT` = 0, or `CAN1_PORT` = 1

*errors*

Pointer to a **CAN_ERRORS** data structure that will be written to by this API to report the error count and error status information for the specified CAN port.

## Return Value

**Error** code -
`CAN_EC_SUCCESS` = 0: Success
`CAN_EC_INVALID_PARAMS` = -1: `port_idx` invalid or `errors` pointer invalid

## Comments

This function reports transmitter/receiver error counts as specified by the Bosch CAN specification. Also some discrete bits are reported which indicate the bus error state.

Inside this API the interrupts are masked for a short period and then re-enabled. This function is reentrant.

## See Also

**CAN_Control** **CAN_Open_Port** **CAN_Close_Port**
**CAN_Status**

## RTOS API

This library function uses a **dynamic** link to reach the **service** offered by RTOS software **interrupt**.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## FD_Next

Iterator for sets of sockets.

```
int FD_Next ( const fd_set * fds_set,
              int previous_fds );
```

## Parameters

*fds_set*

Pointer to the **fd_set** bit field whose sockets are bing iterated.

*previous_fds*

One count greater than the bit number in the fds_set->fds_bits[] bit field at which to start the scan. (This argument serves as the iterator's cursor.) The bit scan sequences from high indexes down to zero.

## Return Value

If greater or equal zero: Next socket descriptor
Negative: End of list (no more sockets)

## Comments

This library function can be used to iterate through the set of sockets created by a **select()** (or **select_ms()**) call.

On first call, the `previous_fds` should be set one count greater than the highest socket descriptor being handled. This would normally be the same value as had been passed to the **select()** API as the `nfds` parameter. Then on subsequent calls to this socket iterator, the `previous_fds` should be set to the previous return value from this function.

When a negative value is returned, all sockets have been interated and the end of list has been reached.

## See Also

**select** **select_ms**

## RTOS API

This library function is implemented without any calls into the @CHIP-RTOS.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| any | any | any | any | any |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## IPsec_Restore_Policy

Restore IP security policy and/or preshared keys from file.

```
int IPsec_Restore_Policy(const char *filename) ;
```

## Parameters

*filename*

Path to binary file containing IP security policy and preshared keys.

## Return Value

Zero on success
On failure non-zero indicates error:

-3 : Policy file exceeded 64 Kbyte size limit
-4 : Work buffer memory allocation failed
-5 : Invalid policy file contents
202 : Policy file open failed
Between 1000 and 1999 :
Policy add operation failed. Subtract 1000 from this error code and look **here**for resulting error code description.
Between 2000 and 2999:
Preshared key add operation failed. Subtract 2000 from this error code and look **here**for resulting error code description.

## Comments

The IP security must be **started** prior to using this API.

This API allows IP security policy and preshared keys to be set based on a stored binary file's contents. The file can be created with the **IPsec_Store_Policy()** library function, or other tools conforming to the **specified** format.

The policies from the file are appended to any which are already in affect. The **IPsec_Clear_Policy()** can be used before this API for a fresh start.

An alternative way to load policy from file is to use the **POLICY_FILE** entry in the CHIP.INI file.

## See Also

**IPsec_Add_Policy()** **IPsec_Store_Policy()**
**IPsec_Clear_Policy()**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **IPSEC_VECTORS** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## IPsec_Status

Read out the status of the IP security function within the system's TCP/IP stack.

```
int IPsec_Status(int *error_code) ;
```

## Parameters

*error_code*

Optional output parameter is the **error** code reported when IP Security initialization was not successful. Set this parameter to NULL if not interested.

## Return Value

Enumerator:

IPSEC_OFF (0) - Not **started**yet.
IPSEC_FAILED (1) - Startup failed, perhaps insufficient **memory**.
IPSEC_READY (2) - Ready for operation.

## Comments

The optional error code value returned to `error_code` will be non-zero if the reported state is `IPSEC_FAILED`.

## See Also

**IPsec_Start()** **IPsec_IKE_Status()**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **IPSEC_VECTORS** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# IP Security

---

Header File: `#include <IPsec.H>`

## Configuration Functions

| | |
|---|---|
| IPsec_Start | Start IP Security running |
| IPsec_Set_Option | Set IPsec/IKE Options |
| IPsec_Add_Policy | Add Policies |
| IPsec_Restore_Policy | Read Policy file |
| IPsec_Store_Policy | Write Policy file |
| IPsec_Clear_Policy | Clear Policies |
| IPsec_Status | Check state |
| IPsec_IKE_Status | Check IKE state |

## IKE Functions

| | |
|---|---|
| IPsec_IKE_Phase1_Set | Add a phase 1 transform |
| IPsec_IKE_Phase1_Delete | Remove a phase 1 transform |

## Preshared Key Functions

| | |
|---|---|
| IPsec_Add_Preshared_Key | Add a preshared key |
| IPsec_Remove_Preshared_Key | Remove a preshared key |
| IPsec_Clear_Preshared_Keys | Clear out all preshared keys |

IP Security over view
IP Security data structures
IP Security/IKE error codes
Known problems with implementation.

List of C Libraries
@CHIP-RTOS Main Index

## BIOS_Set_Batch_Mode

Set **batch file** mode.

```
void BIOS_Set_Batch_Mode ( BATCH_MODE mode,
                           unsigned int enable_timeout );
```

## Parameters

*mode*

BATCH_CONCURRENT= 0: Sets BATCHMODE =0 = concurrent
BATCH_SEQUENTIAL= 1: Sets BATCHMODE =1 = sequential

*enable_timeout*

Boolean:
0: Disable the maximum delayed execution time-out of DOS programs
1: Enable the maximum delayed execution time-out of DOS programs in a batchfile, if BATCHMODE = 1 (sequential)

## Return Value

-- none --

## Comments

This function sets the batch file execution mode of DOS programs for either concurrent or sequential execution. See BATCHMODE initialization **documentation** for details.

The enable_timeout parameter only applies to BATCH_SEQUENTIAL mode.
**Important:**

If BATCHMODE=1take 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 **BIOS_Batch_Continue**API function, which immediately enables 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 enable_timeoutparameter is set to 0, the successor program in a batch file waits forever for execution, if the predecessor program does not finish or call **BIOS_Batch_Continue**.

## See Also

**BIOS_Batch_Continue**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS C Library - BIOS API

Header File: `#include <BIOS_API.H>`

stdio

Files

Ethernet/PPP

IPC@CHIP®

## stdio Functions

| | |
|---|---|
| BIOS_kbhit | Test if character is ready on standard input. |
| BIOS_getch | Get character from standard input, with wait. |
| BIOS_putch | Output character on standard output. |
| BIOS_Set_Focus | Select stdio focus. |
| BIOS_Set_Focus_Hotkey | Set the stdio focus key. |
| BIOS_Select_Stdio | Specify stdin/stdout channels. |
| BIOS_Get_Stdio | Check stdin/stdout channels. |
| BIOS_Install_User_Stdio | Install user stdio handlers. |

## Files Functions

| | |
|---|---|
| BIOS_Get_File_Sharing_Mode | Check the file sharing mode. |
| BIOS_Set_File_Sharing_Mode | Enable/Disable file sharing. |
| BIOS_Fast_Findfirst | Fast findfirst. |
| BIOS_Fast_Findnext | Fast findnext. |
| BIOS_Fast_Find_Done | Unlock directory after search. |
| BIOS_Get_File_Name | Get file name by handle. |
| BIOS_LFN_Findfirst | Long file name findfirst. |
| BIOS_LFN_Findnext | Long file name findnext. |
| BIOS_LFN_Find_Done | Unlock directory after search. |
| BIOS_LFN_Get_Name | Get long file name by handle. |

| | |
|---|---|
| BIOS_Disk_Info | Report disk statistics. |
| BIOS_Disk_Space | Report disk bytes free. |

## Ethernet/PPP Functions

| | |
|---|---|
| BIOS_Get_MAC | Get MAC address of the Ethernet interface. |
| BIOS_Get_MAC2 | Get MAC address of the second ethernet interface. |
| BIOS_Config_Server_Level | Change level for configuration server. |
| BIOS_Server_On_Off | Suspend/Resume system servers. |
| BIOS_Ethernet_State | Detect Ethernet link state. |
| BIOS_Connection_Handler | Install a system server connection handler. |
| BIOS_Install_UDP_Cfg_Callback | Install a UDP Config Server user callback. |
| BIOS_Get_PPPServer_IP | Get the IP address of the PPP server. |
| BIOS_Get_PPPClient_IP | Get the IP address of the PPP client. |

## IPC@CHIP® Functions

| | |
|---|---|
| BIOS_Get_Serial_Number | Get the serial number of the IPC@CHIP®. |
| BIOS_Get_Product_Serial_Number | Get the serial number of the BECK product. |
| BIOS_Get_Hardware_Rev | Get the hardware revision number of the IPC@CHIP® device. |
| BIOS_Get_Product_Hardware_Rev | Get the hardware revision number of the BECK product. |
| BIOS_Get_Boot_Version | Get bootstrap version number. |
| BIOS_Get_RTOS_Version | Get RTOS version. |
| BIOS_Version_String | Get RTOS version string. |
| BIOS_Execute | Execute a command shell command. |
| BIOS_Set_Timer_0x1C | Set timer 0x1C interval. |
| BIOS_Set_Timer_0xAF | Set timer 0xAF interval. |
| BIOS_Set_Batch_Mode | Set Batch file mode. |
| BIOS_Batch_Continue | Continue batch file execution. |
| BIOS_Get_Features | Report RTOS features. |
| BIOS_Power_Save | Enter low power standby mode. |
| BIOS_Power_SaveEx | Make power save mode settings. |
| BIOS_Install_Error_Handler | Install a user fatal error handler. |
| BIOS_Reboot | Reboot the CPU. |
| BIOS_Set_Ini_String | Insert an entry into CHIP.INI. |
| BIOS_Get_Ini_String | Get an entry from CHIP.INI. |
| BIOS_Get_Device_Names | Get the IPC@CHIP® device names. |
| BIOS_Get_Device_NamesEx | Get the IPC@CHIP® device names. (Extended) |
| BIOS_Set_Memory_Gap | Set a memory gap between DOS programs. |

| | |
|---|---|
| **BIOS_Write_Persistent_Data** | Writes persistent data into the @CHIPs Flash memory |
| **BIOS_Read_Persistent_Data** | Reads persistent data from the @CHIPs Flash memory. |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## AddDefaultGateway

Add the default gateway.

```
int AddDefaultGateway ( IP_INTERFACE which_interface,
                        const unsigned long far *Gateway,
                        int *error );
```

This function is **deprecated**! Use **AddDefaultGatewayEx** instead.

## Parameters

*which_interface*

Enumeration type:
- II_ETHERNET = 0: Ethernet
- II_PPP_SERVER = 1: PPP server
- II_PPP_CLIENT = 2: PPP client

*Gateway*

Pointer to gateway IP address in network byte order.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## Comments

If this function is used, the **gateway** entry in the `CHIP.INI` becomes invalidated, but unchanged. If the PPP server or PPP client is specified by the `which_interface` parameter, then the gateway is set to the remote peer IP address.

## See Also

**AddDefaultGatewayEx** **DelDefaultGateway** **GetDefaultGateway**
**AddStaticRoute** **DelStaticRoute**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

IP **Gateway** `CHIP.INI` Configuration
PPP server **Gateway** `CHIP.INI` Configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## DelDefaultGateway

Delete the default gateway.

```
int DelDefaultGateway ( const unsigned long far *Gateway,
                        int *error );
```

## Parameters

*Gateway*

Pointer to gateway IP address to be removed.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## Comments

If this function is used, the **gateway** entry in the `CHIP.INI` becomes invalidated, but unchanged.

## See Also

**AddDefaultGateway** **GetDefaultGateway**
**AddStaticRoute** **DelStaticRoute**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

IP **Gateway** `CHIP.INI` Configuration
PPP server **Gateway** `CHIP.INI` Configuration

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## GetDefaultGateway

Get the current default gateway.

```
int GetDefaultGateway ( unsigned long far *Gateway,
                        int *error );
```

## Parameters

*Gateway*

Output parameter: Pointer to where the gateway IP will be written in network byte order.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## See Also

**AddDefaultGateway** **DelDefaultGateway**
**AddStaticRoute** **DelStaticRoute**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## AddStaticRoute

Add a route for a interface.

```
int AddStaticRoute ( IP_INTERFACE which_interface,
                     const Route_Entry far *route,
                     int *error );
```

## Parameters

*which_interface*

Enumeration type:
- II_ETHERNET = 0: Ethernet
- II_PPP_SERVER = 1: PPP server
- II_PPP_CLIENT = 2: PPP client

*route*

Pointer to user `Route_Entry` **structure**

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## Comments

The `Route_Entry` structure is defined in tcpipapi.h:

Refer to this RTOS API function's **documentation** for more details.

## See Also

**DelStaticRoute**
**AddDefaultGateway** **DelDefaultGateway** **GetDefaultGateway**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## DelStaticRoute

Delete a route from an interface.

```
int DelStaticRoute ( IP_INTERFACE which_interface,
                     const Route_Entry far *route,
                     int *error );
```

## Parameters

*which_interface*

Enumeration type:
- II_ETHERNET = 0: Ethernet
- II_PPP_SERVER = 1: PPP server
- II_PPP_CLIENT = 2: PPP client

*route*

Pointer to user `Route_Entry` **structure**

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## Comments

Only `route` structure members `destIPAddress` and `destNetmask` must be set valid before calling.

## See Also

**AddStaticRoute**
**AddDefaultGateway** **DelDefaultGateway** **GetDefaultGateway**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## IPsec_Clear_Policy

This function clears the list of policies in affect for the IP Security.

```
int IPsec_Clear_Policy(void) ;
```

## Parameters

*-- none --*

## Return Value

**Error code**, zero on success

## Comments

The IP security must be **started** prior to using this API.

A default policy of BYPASS for all IP addresses will be used following this action.

## See Also

**IPsec_Add_Policy()** **IPsec_Set_Option()**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **IPSEC_VECTORS** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## IPsec_Remove_Preshared_Key

This function removes any preshared key that had been established for the specified remote IP.

```
int IPsec_Remove_Preshared_Key(
                      const char *remote_ip) ;
```

## Parameters

*remote_ip*

Zero terminated ASCII string specifying the IP address of the remote side. This string must match what was used when adding this key.

## Return Value

**Error code**, zero on success

## Comments

The IP security must be **started** prior to using this API.

The IP address may be either IPv4 of IPv6. Domain names must be **resolved** first by the caller and are not supported by this API.

## See Also

**IPsec_Add_Preshared_Key()** **getHostByName()**
**IPsec_Clear_Preshared_Keys()**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **IPSEC_VECTORS** for more details.

## Related Topics

**PRESHARED_KEY** definition in CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## IPsec_Clear_Preshared_Keys

Clear all preshared keys from the database.

```
void IPsec_Clear_Preshared_Keys(void) ;
```

## Parameters

*-- none --*

## Return Value

-- none --

## See Also

IPsec_Add_Preshared_Key()
IPsec_Remove_Preshared_Key()

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function IPSEC_VECTORS for more details.

## Related Topics

IPSEC ENABLE from CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## IPsec_IKE_Phase1_Delete

Remove one of the IKE phase 1 transforms.

```
int IPsec_IKE_Phase1_Delete(int index) ;
```

## Parameters

*index*

Index 0 or 1 selects which of two phase 1 IKE transforms is to be deleted.

## Return Value

**error code**, zero on success.

## Comments

This function removes one of the two IKE phase 1 transforms which the user can set. If both user settable phase 1 transforms are not present, then the **default** transform will be used.

## See Also

**IPsec_IKE_Phase1_Set()**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to the RTOS API function **IPSEC_VECTORS** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RTX_GetFineTick

Get real-time interrupt count and hardware TIMER2 up count.

```
void RTX_GetFineTick (RTX_FINE_TICK *report);
```

## Parameters

*report*

Output parameter, pointer to a `RTX_FINE_TICK` structure which will receive the timing information. This structure consists of a 32 bit `Rti_Count` union allowing WORD wise access to the 32 bit Real-Time Interrupt count, followed by a 16 bit TIMER2 reading. See type definition below in comments.

## Return Value

-- none --

## Comments

This procedure's main purpose in life is to support the more user friendly **RTX_GetTick_us** API. However some real-time appplications may be interested in using this function directly to avoid the additional computation overhead added by the conversion to microseconds of the data collected here. For example if an Interrupt Service Routine (ISR) needs to know precisely what time it is then it might be a wiser design decision to buffer the raw data collected by this API for later off-line processing rather than hanging around waiting for a conversion to microseconds within the ISR. (On the other hand, the conversion to microseconds has a data reduction aspect to it.)

If the **RTX_FineTickInit** API has not yet been called, then this call will be made one time to obtain the `_pRti_Tick` internal data **pointer** that is required here. Making this additional API call is time consuming (slightly) and results in the interrupts being enabled. Therefore calling **RTX_FineTickInit** prior to using this API may be advisable.

Once this initialization has been handled, this API is well suited for real-time applications. It becomes both re-entrant and respects the interrupt masking applied by the caller. Consequently calling this API from inside hardware interrupts is no problem. These interrupt service routines do not need to be **registered** with the RTX. No further calls into the @Chip-RTOS are necessary since both the @Chip-RTOS 32 bit RTI up counter (via the pointer) and the TIMER2 hardware count are sampled here directly.

The `_RTI_kHz` global **constant** can be inspected to determine the **RTI** rate in effect (always 1 for SC1x systems).

The data returned by this API is output to the following structure type.

```
typedef struct {
    union {
        unsigned long   Full ;      // 32 bit RTI counter
        struct {
            unsigned int Lsh ;      // Least Significant Half
            unsigned int Msh ;      // Most Significant Half
        } Part ;
    } Rti_Count ;

    unsigned int    Timer2_Count ;

} RTX_FINE_TICK ;
```

## See Also

RTX_FineTickInit  RTX_GetTick_us
RTX_Get_RTI_Ticks  RTX_GetTickCount

## RTOS API

After RTX_FineTickInit has been invoked a single time, this library function makes no entry into the RTOS.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---

## BIOS_kbhit

Test if character is ready at standard input(s).

```
unsigned int BIOS_kbhit ( void );
```

## Parameters

-- none --

## Return Value

Boolean: '1' if characters is ready, '0' if no character ready.

## Comments

All devices selected as standard input (e.g. EXT, COM and Telnet) are polled for data ready.

## See Also

BIOS_getch BIOS_putch BIOS_Set_Focus
BIOS_Set_Focus_Hotkey BIOS_Select_Stdio BIOS_Get_Stdio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## BIOS_getch

Get char from standard input, with wait.

```
unsigned int BIOS_getch ( void );
```

## Parameters

-- none --

## Return Value

Lower byte: input character
Upper byte: Source of input - 1: EXT, 2: COM, 4: Telnet, 8: User channel

## Comments

If only the input character is of interest, the return value can be cast to a character to suppress the data source indication carried in the upper byte.

## See Also

BIOS_kbhit BIOS_putch BIOS_Set_Focus
BIOS_Set_Focus_Hotkey BIOS_Select_Stdio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## BIOS_putch

Output character to standard output(s).

```
void BIOS_putch ( unsigned char output );
```

## Parameters

*output*

Character to be output

## Return Value

-- none --

## Comments

This character is output to each device selected for standard output (e.g. EXT, COM and Telnet).

## See Also

BIOS_getch BIOS_kbhit BIOS_Set_Focus
BIOS_Set_Focus_Hotkey BIOS_Select_Stdio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## BIOS_Set_Focus

Set STDIO focus.

```
void BIOS_Set_Focus ( IO_FOCUS focus );
```

## Parameters

*focus*

Enumeration type:
FOCUS_SHELL = 1: Set focus to command shell
FOCUS_APPLICATION = 2: Set focus to application
FOCUS_BOTH = 3: Set focus to both application and command shell

## Return Value

-- none --

## Comments

This function sets the focus of STDIO to either console, application or both.

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.

This function includes a one millisecond sleep.

## See Also

BIOS_Set_Focus_Hotkey BIOS_getch BIOS_putch
BIOS_kbhit BIOS_Select_Stdio BIOS_Get_Stdio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Related Topics

**Focus** key definition

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Set_Focus_Hotkey

Set the stdio focus key.

```
void BIOS_Set_Focus_Hotkey ( unsigned char hotkey );
```

## Parameters

*hotkey*

New focus key code in range: [ 0 .. 254 ]

## Return Value

-- none --

## Comments

The default hot key is CTRL-F, ASCII 6.

If the focus key is set to zero, the switching of stdio from the console is disabled.

At runtime, the hot key (e.g. Ctrl-F) toggles the stdio between three modes:

Stdio: Shell - Focus set to command shell
Stdio: User - Focus set to applications
Stdio: Both - Command shell and applications

The focus key is not usable by the command shell or DOS executable.

## See Also

BIOS_Set_Focus BIOS_getch BIOS_putch
BIOS_kbhit BIOS_Select_Stdio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Related Topics

Focus key definition in CHIP.INI.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Select_Stdio

Specify the stdin / stdout channels.

```
int BIOS_Select_Stdio ( unsigned int ports, unsigned char direction );
```

## Parameters

*ports*

Bit field specifying the channels using a combination of following masks:

B0: STDIO_EXT - EXT serial port
B1: STDIO_COM - COM serial port
B2: STDIO_TELNET - Telnet
B3: STDIO_USER - User port
B4: STDIO_SSH - SSH (SC1x3/SC23 only)

*direction*

Bit field specifying either stdin, stdout or both using a combination of following masks

B0: SET_STDOUT
B1: SET_STDIN

## Return Value

0: Successful
-1: Invalid `paragraphs`

## Comments

The bit masks are defined in BIOS_API.H include file.

## See Also

BIOS_Get_Stdio BIOS_kbhit BIOS_putch
BIOS_Set_Focus BIOS_Set_Focus_Hotkey BIOS_getch
BIOS_Install_User_Stdio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V1.07 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Get_Stdio

Get the stdin / stdout channels setting.

```
unsigned int BIOS_Get_Stdio ( void );
```

## Parameters

-- none --

## Return Value

Two bit fields, each one byte. LSB specifies the stdin, MSB specifies the stdout.

## Comments

Both bit fields returned (MSB for stdout, LSB for stdin) are coded as follows:

B0: STDIO_EXT - EXT serial port
B1: STDIO_COM - COM serial port
B2: STDIO_TELNET - Telnet
B3: STDIO_USER - User port
B4: STDIO_SSH - SSH (SC1x3/SC23 only)

The bit masks are defined in BIOS_API.H include file.

## See Also

BIOS_Select_Stdio BIOS_kbhit BIOS_putch
BIOS_Set_Focus BIOS_Set_Focus_Hotkey BIOS_getch

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V1.07 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Install_User_Stdio

Install user specific stdio handlers.

```
void BIOS_Install_User_Stdio ( const USER_STDIO_FUNCS far *vectors );
```

## Parameters

*vectors*

Pointer to a USER_STDIO_FUNCS type data structure prepared by caller. Set this parameter to null to uninstall the user's device handlers.

## Return Value

-- none --

## Comments

This function installs user specific stdio channel handlers, allowing the user to connect their own input/output device such as a display or keyboard to the IPC@CHIP®. A TCP application similar to Telnet would be another example.

The user must implement four functions in their application to cover reading and writing characters from/to their new stdin/stdout device. After installing these functions with this API call and then setting stdin and/or stdout to the new user channel with the **BIOS_Select_Stdio** API, the @Chip-RTOS will call these user functions at each stdin and stdout operation.

The following type definitions can be found in BIOS_API.H header file:

```
typedef int huge (far *USER_KBHIT) (void);
typedef void huge(far *USER_PUTCH) (char chr);
typedef void huge(far *USER_PUTSTR) (const char far * pch, int n);
typedef int huge (far *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 be implemented by the user:

```
// user_kbhit returns a 1 if a character is available, 0 if not.
int huge user_kbhit(void);

// user_getch reads a char from stdin (waits if none available)
int huge user_getch(void);

// user_putch writes a single character to stdout
void huge user_putch(char chr);     //write a single char to stdout

// user_putstr writes a string of n characters to stdout
void huge user_putstr(const char far * pch, int n);
```

The user must set the vectors to this set of four callback functions in the USER_STDIO_FUNCS structure referenced by the `vectors` input parameter.

The user handlers are uninstalled by calling this API with the `vectors` parameter set to zero.

**Important** :

1. The USER_STDIO_FUNCS data structure must be static. The @Chip-RTOS does not make a copy of this information, but instead references this provided data structure during operation.
2. If your application exits, don't forget to uninstall your stdio handler by calling this API call with `vectors`parameter set to zero.
3. Do not call your stdio functions from within your application, these functions must only be called from the @Chip-RTOS.

Example (Borland C):

```
USER_STDIO_FUNCS user_stdio_funcs ;    // global variable

// implementations of user's stdio functions
int  huge my_kbhit(void)
{
   //........
}

int huge my_getch(void)
{
   //........
}

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

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

void install_my_stdio_channel (void)
{
   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;

   BIOS_Install_User_Stdio ( &user_stdio ) ;
   BIOS_Select_Stdio ( STDIO_COM | STDIO_TELNET | STDIO_USER, // ports
           SET_STDOUT | SET_STDIN) ;              // direction
}

void remove_my_stdio_channel (void)
{
   BIOS_Select_Stdio ( STDIO_COM | STDIO_TELNET,              // ports
           SET_STDOUT | SET_STDIN) ;              // direction
```

```
   BIOS_Install_User_Stdio ( 0 ) ;             // un-install
}

int main (void)
{
   //......
   install_my_stdio_channel();
   //....

   // at the end of the program
   remove_my_stdio_channel():
}
```

## See Also

BIOS_Select_Stdio  BIOS_kbhit  BIOS_putch
BIOS_Set_Focus  BIOS_Set_Focus_Hotkey  BIOS_getch
BIOS_Get_Stdio

## RTOS API

This library function invokes a RTOS software interrupt.  Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Get_File_Sharing_Mode

Get the current file sharing mode.

```
int BIOS_Get_File_Sharing_Mode ( void );
```

## Parameters

-- none --

## Return Value

sharing mode: Non-zero if sharing mode is enabled

## Comments

By default file sharing 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. To avoid this security feature you can enable file sharing. This can also be done with the CHIP.INI entry **FILESHARING**.

## See Also

**BIOS_Set_File_Sharing_Mode**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## BIOS_Set_File_Sharing_Mode

Get the current file sharing mode.

```
void BIOS_Set_File_Sharing_Mode ( unsigned int enable_sharing );
```

## Parameters

*enable_sharing*

Boolean: Set non-zero to enable file sharing, or zero to disable file sharing.

## Return Value

-- none --

## Comments

By default file sharing 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. To avoid this security feature you can enable file sharing. This can also be done with the CHIP.INI entry **FILESHARING**.

## See Also

**BIOS_Get_File_Sharing_Mode**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## BIOS_Fast_Findfirst

Provide an access to the file system directories which locks the directory against write activity during the directory search.

```
int BIOS_Fast_Findfirst (const char far *filename,
                         FILE_FIND far *find,
                         unsigned int attributes );
```

## Parameters

*filename*

Pointer to null terminated ASCII file name pattern for search which may contain * and ? wild card characters.

*find*

Pointer to a **FILE_FIND** structure that will be initialized inside this API call.

*attributes*

File attributes (provisional, not used!)

## Return Value

0: No file found
1: Success

## Comments

This function and the **BIOS_Fast_Findnext** provide a slightly faster findfirst/next access than do the DOS compatible functions available on software interrupt **0x21**. (Historically, this function was significantly faster than the @Chip-RTOS's DOS findfirst/findnext implementation. But since this time the DOS functions have been corrected so the speed advantage is no longer an issue here.) More importantly, this function places a write protection lock on the directory being searched for use in a mult-tasking environment.

This function is reentrant. However, eventually the system will run out of file system resources (e.g. directory nodes) if too many sessions are simultaneously active.

These functions work in a similar way to the DOS SWI 0x21 findfirst/next functions. BIOS_Fast_Findfirst must be called first. Then the BIOS_Fast_Findnext is used repeatedly as required to iterate through all the files in a directory matching the specified `filename` pattern.

When a lock protection against directory writes is not desired between calls to the file iteration functions,

the normal C-library findfirst/findnext functions should instead be used.

**CAUTION:**

The **BIOS_Fast_Find_Done** function must be called if the Findfirst/Findnext sequence is stopped prior to these iteration functions returning with a zero ("File not found") indication. This is important so that the directory can be unlocked and file system resources released!

## SC1x3/SC23 Comments

Long file names can be accessed with the **BIOS_LFN_Findfirst** function.

## See Also

**BIOS_Fast_Findnext** **BIOS_Fast_Find_Done** **BIOS_LFN_Findfirst**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Fast_Findnext

Continue iteration of files in a directory locked against write activity.

```
int BIOS_Fast_Findnext ( FILE_FIND far *find );
```

## Parameters

*find*

Pointer to the **FILE_FIND** structure initialized by a **BIOS_Fast_Findfirst** call.

## Return Value

0: No file found
1: Success

## Comments

This function continues the iteration of files found in a directory. The directory search process must be started with the **BIOS_Fast_Findfirst** function.

This function is reentrant.

**Important:**

The **BIOS_Fast_Find_Done**function must be called when finished with the iteration to unlock the directory unless a zero "No file found" indication was returned on your final call to `BIOS_Fast_Findnext`. (The resources have already been released in this case.)

## See Also

**BIOS_Fast_Findfirst** **BIOS_Fast_Find_Done**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.20** | **V1.20** | **V1.20** | **V0.91** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Fast_Find_Done

Unlock the directory following Findfirst/next operation.

```
int BIOS_Fast_Find_Done ( FILE_FIND far *find );
```

## Parameters

*find*

Pointer to the FILE_FIND structure used at the **BIOS_Fast_Findfirst** call.

## Return Value

0: Successful
-1: Findfirst not active

## Comments

The function unlocks the directory being searched by the **BIOS_Fast_Findfirst** and **BIOS_Fast_Findnext** functions.

This function is reentrant.

## See Also

**BIOS_Fast_Findfirst** **BIOS_Fast_Findnext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

End of document

## BIOS_Get_File_Name

Get file name by handle.

```
int BIOS_Get_File_Name ( char far * filename,
                         unsigned int handle );
```

## Parameters

*filename*

Output parameter: Pointer to 13 character buffer which will receive the nil terminated file name associated with the provided handle.

*handle*

File handle.

## Return Value

0: Success
-1: Invalid file handle

## SC1x3/SC23 Comments

The alternative **BIOS_LFN_Get_Name** function can be used to retrieve the long file names.

## See Also

**BIOS_LFN_Get_Name**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_LFN_Findfirst

Begin iteration of files in a specified long file name path.

```
int BIOS_LFN_Findfirst (const char far *filename,
                        LFN_FILE_FIND far *find),
                        unsigned char     lock_the_node);
```

## Parameters

*filename*

Pointer to null terminated ASCII file name pattern for search which may contain * and ? wild card characters.

*find*

Pointer to a **LFN_FILE_FIND** structure that will be initialized inside this API call.

*lock_the_node*

Boolean set non-zero to leave the directory node lock protected against write on successful return. Set to zero to leave the directory unlocked on return.

## Return Value

0: No file found
1: Success

## Comments

This function and the **BIOS_LFN_Findnext** provide access to the long file names. When the `lock_the_node` Boolean is FALSE then these functions behave similar to the DOS findfirst/findnext functions provided by the compiler C-libraries. A significant difference is that the **BIOS_LFN_Find_Done** function must be called to prevent a memory leak if the search is terminated before the end.

When the `lock_the_node` Boolean is TRUE then these functions behave similar to the **BIOS_Fast_Findfirst** and **BIOS_Fast_Findnext** functions. A write protection lock is placed on the directory being scanned by these functions.

This function is reentrant. However, eventually the system will run out of file system resources (e.g. directory nodes) if too many sessions are simultaneously active.

**CAUTION:**

The **BIOS_LFN_Find_Done**function must be called if the Findfirst/Findnext sequence is stopped prior to these iteration functions returning with a zero ("File not found") indication. This is important so that the directory can be unlocked and system resources released!

## See Also

**BIOS_LFN_Findnext** **BIOS_LFN_Find_Done** **BIOS_Fast_Findfirst**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_LFN_Findnext

Continue iteration of files in a directory.

```
int BIOS_LFN_Findnext ( LFN_FILE_FIND far *find );
```

## Parameters

*find*

Pointer to the **LFN_FILE_FIND** structure initialized by a **BIOS_LFN_Findfirst** call.

## Return Value

0: No file found
1: Success

## Comments

This function continues the iteration of files found in a directory. The directory search process must be started with the **BIOS_LFN_Findfirst** function.

This function is reentrant.

**Important:**
The **BIOS_LFN_Find_Done**function must be called when finished with the iteration to unlock the directory unless a zero "No file found" indication was returned on your final call to `BIOS_LFN_Findnext`. (The resources have already been released in this case.)

## See Also

**BIOS_LFN_Findfirst** **BIOS_LFN_Find_Done**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_LFN_Find_Done

Unlock the directory following Findfirst/next operation.

```
void BIOS_LFN_Find_Done ( LFN_FILE_FIND far *find );
```

## Parameters

*find*

Pointer to the **LFN_FILE_FIND** structure initialized by a **BIOS_LFN_Findfirst** call.

## Return Value

-- none --

## Comments

The function release memory resources and unlocks the directory being searched by the **BIOS_LFN_Findfirst** and **BIOS_LFN_Findnext** functions.

This function is reentrant.

## See Also

**BIOS_LFN_Findfirst** **BIOS_LFN_Findnext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|

| V2.02 |
| --- |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_LFN_Get_Name

Get long file name by handle.

```
int BIOS_LFN_Get_Name ( char far * filename,
                        unsigned int handle );
```

## Parameters

*filename*

Output parameter: Pointer to 260 character (LFN_MAXPATH) buffer which will receive the nil terminated long file name associated with the provided handle.

*handle*

File handle.

## Return Value

0: Success
-1: Invalid file handle

## Comments

The **BIOS_Get_File_Name** function can be used to retrieve the short alias DOS file names.

## See Also

**BIOS_Get_File_Name**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Get_MAC

Get MAC address of the Ethernet interface.

```
void BIOS_Get_MAC ( unsigned char far *mac );
```

## Parameters

*mac*

Output parameter: Pointer to a 6 byte memory area where the MAC address will be stored

## Return Value

-- none --

## Comments

This function outputs the Ethernet MAC address as a six byte array.

## See Also

Get_IPConfig

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## BIOS_Get_MAC2

Get MAC address of the second ethernet device.

```
void BIOS_Get_MAC2 ( unsigned char far *mac );
```

## Parameters

*mac*

Output parameter: Pointer to a 6 byte memory area where the MAC address will be stored

## Return Value

-- none --

## Comments

This function outputs the Ethernet MAC address as a six byte array.

The second ethernet controller has no phy controller. An MII interface is provided. It is only accessible with an external connected phy and an external driver program, which must base on the **TCPIP device driver API**.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## BIOS_Config_Server_Level

Change level for configuration server.

```
void BIOS_Config_Server_Level ( unsigned int level );
```

## Parameters

*level*

The supported level

## Return Value

-- none --

## Comments

This function changes the supported level for the configuration server. For a description of the possible levels, refer to **UDPCFG** level topic.

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 has no effect. To avoid this, use an unlisted support level such as 0x1000 in the `CHIP.INI`.

The entry in the `CHIP.INI` file is not changed by this call.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## BIOS_Ethernet_State

Detect Ethernet link state.

```
unsigned int BIOS_Ethernet_State ( unsigned int *init_error,
                                   unsigned int *phy_state );
```

## Parameters

*init_error*

non-zero indicates initialization or reset procedure of Ethernet device failed.

*phy_state*

SC13/SC1x3/SC23 only: phy status
Bit 14 1 = Link Not detected
Bit 07 1 = 100Base-TX mode, 0=10Base-T mode
Bit 06 1 = Device in Full Duplex mode

## Return Value

non-zero indicates have no Ethernet link.

## Comments

If return value is non-zero (no Ethernet link), this may be due to the network cable not being connected.
In the SC12, a zero is output to `phy_state`.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**

End of document

## BIOS_Connection_Handler

Install a system server connection handler function.

```
int BIOS_Connection_Handler ( CONN_HANDLER handler,
                              SERVER_SPEC which_server );
```

## Parameters

*handler*

Vector to callback function

*which_server*

Enumerator:
SS_FTP_SERVER = 0
SS_TELNET_SERVER = 1
SS_WEB_SERVER = 2

SC1x3/SC23 only:
SS_SSL_WEB_SERVER = 3
SS_SSH_SERVER = 4

## Return Value

0: Success
-1: Invalid `which_server` parameter

## Comments

The installed connection handler will be executed by TCPIP if a client establishes a connection to the server (FTP, WEB, Telnet,SSL Webserver, SSH Server).

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

```
int huge UserConnectionHandler(
                      struct sockaddr far *sockptr );
```

Example usage:

The implemented handler function could check the source IP address (Client's IP), comparing this IP with an application internal list of allowed IP addresses and reject the connection by returning a non-zero

value if the source IP is not in the list.

If the connection handler returns zero the connection will be established. If it returns a non-zero value, the connection will be aborted.
A connection handler can be removed by calling `BIOS_Connection_Handler` with a null pointer for the `handler` parameter.

## SC1x Comments

Example handler:
The handler can read the IP Address and the Port in the **sockaddr_in struct** . (If desired, the IP address in the `sin_addr` structure member can be converted to ASCII using the TCP/IP **API_INETTOASCII** function.)

```
int huge UserConnectionHandler( struct sockaddr * SockAddr )
{
    char ip_str[INET_ADDRSTRLEN];
    InetToAscii(&((struct sockaddr_in *)SockAddr)->sin_addr.s_addr
                ,ip_str);
    helper_printf("\r\nHost %s, Port %u",ip_str,
                ((struct sockaddr_in *)SockAddr)->sin_port);
    return 0;
}
```

## SC1x3/SC23 Comments

Example handler:
The handler can read the IP Address and the Port in the **sockaddr_storage struct** . Both address modes IPv4 and IPv6 are supported.

```
int huge UserConnectionHandler( struct sockaddr * SockAddr )
{
  char ip_str[INET6_ADDRSTRLEN];
  struct sockaddr_in ipv4SockAddr;

  if(IN6_IS_ADDR_V4MAPPED(
      &((struct sockaddr_storage *)SockAddr)->addr.ipv6.sin6_addr))
  {
   //IPv4, copy data into  sockaddr_in variable
   ipv4SockAddr.sin_addr.s_addr=((struct sockaddr_storage *)SockAddr)
                               ->addr.ipv6.sin6_addr.ip6Addr.ip6U32[3];
   ipv4SockAddr.sin_port=
          ((struct sockaddr_storage *)SockAddr)->addr.ipv6.sin6_port;
   //Convert to string
   inet_ntop( AF_INET,&ipv4SockAddr.sin_addr,ip_str,INET_ADDRSTRLEN);

   //helper_printf(.........);
  }
 else
  {
   //IPv6: //Convert to string
   inet_ntop( AF_INET6,
             &((struct sockaddr_storage *)SockAddr)
             ->addr.ipv6.sin6_addr,ip_str,INET6_ADDRSTRLEN);
   //helper_printf(...........);
  }
  return 0;
}
```

## See Also

**InetToAscii** **inet_ntop**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Install_UDP_Cfg_Callback

Install a UDP Config Server user callback function.

```
void BIOS_Install_UDP_Cfg_Callback ( UDP_CFG_CALLBACK callback );
```

## Parameters

*callback*

Vector to callback function

## Return Value

-- none --

## Comments

This function installs a user callback function for the UDP Config Server. This function will be invoked when a UDP configuration request with command number 06 is received. This allows you to implement your own UDP Config sub protocol and commands.

The UDP Cfg callback function must have the following form:

```
void huge MyUdpCfgSrvCB( struct UdpCfgSrv_UserCBInfo far *infoPtr );
```

The UDP Cfg callback function is passed an argument with information about the UDP Cfg Request and its requester. This data structure, defined in BIOS_API.H header file, is repeated below. The **sockaddr_in** structure definition can be found in TCPIPAPI.H

If the callback function sets a pointer in the `dataPtr` member of the `UdpCfgSrv_UserCBInfo` structure, the UDP Config will be sent the referenced data using the `dataLength` member of the structure for the byte count. If this pointer is set to null or the `dataLength` field is set to 0 then no data will be sent back to the requester.

To remove an installed callback function, call this API function with a null pointer.

For more information on the UDP Config Server and its protocol have a look at the UDP Config Server description available on our website.

Note: The amount data sent and received is limited to 300 bytes maximum.

```
typedef struct UdpCfgSrv_UserCBInfo
{
  int    length ;                          // Length of this structure
  struct sockaddr_in far *fromAddrPtr;     // Sender address pointer
  int    udpCfgSD;                         // Socket descriptor from UDP Config Server Socket
  char   far *dataPtr;                     // Data of Request package
  unsigned   dataLength;                   // Length of request package

} UDPCFGSRV_USERCBINFO ;
```

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Get_PPPServer_IP

Get the IP address of the PPP server.

```
void BIOS_Get_PPPServer_IP ( char far *ipaddr );
```

## Parameters

*ipaddr*

Pointer to a 16 byte memory area where the IP address is to be stored as a null terminated string. If no PPP IP address is set (e.g. no link is established) the function return an empty string.

## Return Value

none

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## BIOS_Get_PPPClient_IP

Get the IP address of the PPP client.

```
void BIOS_Get_PPPClient_IP ( char far *ipaddr );
```

## Parameters

*ipaddr*

Pointer to a 16 byte memory area where the IP address is to be stored as a null terminated string. If no PPP IP address is set (e.g. no link is established) the function return an empty string.

## Return Value

none

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document




# @CHIP-RTOS C Library - BIOS API

## BIOS_Get_Serial_Number

Get the serial number of the IPC@CHIP® device.

```
unsigned long BIOS_Get_Serial_Number ( void );
```

## Parameters

-- none --

## Return Value

IPC@CHIP® serial number

## Comments

The IPC@CHIP® serial number is a 24 bit value.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document




# @CHIP-RTOS C Library - BIOS API

---

## BIOS_Get_Product_Serial_Number

Get the serial number of the BECK product.

```
unsigned long BIOS_Get_Product_Serial_Number ( void );
```

## Parameters

-- none --

## Return Value

BECK product serial number

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Get_Hardware_Rev

Get the hardware revision number of the IPC@CHIP® device.

```
unsigned int BIOS_Get_Hardware_Rev ( void );
```

## Parameters

-- none --

## Return Value

IPC@CHIP® hardware revision number

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## BIOS_Get_Product_Hardware_Rev

Get the hardware revision number of the BECK product.

```
unsigned int BIOS_Get_Product_Hardware_Rev ( void );
```

## Parameters

-- none --

## Return Value

BECK product hardware revision number

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## BIOS_Get_Boot_Version

Get bootstrap version number.

```
unsigned int BIOS_Get_Boot_Version ( void );
```

## Parameters

-- none --

## Return Value

Bootstrap program version number, in BCD format:
MSB is major version, LSB is minor version.

## Comments

Example:
If the function returns 0x020C, this means that you have version 2.12.

## See Also

BIOS_Get_RTOS_Version

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## BIOS_Get_RTOS_Version

Get RTOS version number.

```
unsigned int BIOS_Get_RTOS_Version( unsigned int *beta_flag );
```

## Parameters

*beta_flag*

Optional output parameter: If non-null, this location will be set as a Boolean to indicate if the RTOS is a beta version. A zero indicates that it is not a beta version, non-zero indicates a beta version.

## Return Value

RTOS program version number, in split binary format:
MSB is binary major version, LSB is binary minor version.

## Comments

Example:
If the function returns 0x0100, this means that you have RTOS version 1.00.

## See Also

BIOS_Get_Boot_Version  BIOS_Version_String

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## BIOS_Version_String

Get RTOS version string.

```
void BIOS_Version_String ( char far *version, int max_len );
```

## Parameters

*version*

Output parameter: Up to `max_len` characters of the RTOS's version string will be output here.

*max_len*

Maximum length of `version` string to copy (including the binary zero character for terminating the string).

## Return Value

-- none --

## Comments

**Caution:**

The string at `version`must be one greater than `max_len`to accommodate the string terminating nil character which is output after the `version`string.

## See Also

**BIOS_Get_RTOS_Version**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Execute

Execute a command shell command.

```
int BIOS_Execute ( const char far *command_line );
```

## Parameters

*command_line*

Pointer to a null terminated command line.

## Return Value

0 : succes
!=0 : on error

## Comments

This function passes the command string to the command interpreter.

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

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## BIOS_Set_Timer_0x1C

Set timer 0x1C interval.

```
void BIOS_Set_Timer_0x1C ( unsigned int interval_ms );
```

## Parameters

*interval_ms*

Timer interval in milliseconds.

## Return Value

-- none --

## Comments

This function defines the interval in milliseconds that timer interrupt 0x1C will use..

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` .

## See Also

BIOS_Set_Timer_0xAF

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Set_Timer_0xAF

Set timer 0xAF interval.

```
void BIOS_Set_Timer_0xAF ( unsigned int interval_ms );
```

## Parameters

*interval_ms*

Timer interval in milliseconds.

## Return Value

-- none --

## Comments

This function defines the interval in milliseconds that timer interrupt 0xAF will use..

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` .

## See Also

BIOS_Set_Timer_0x1C

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Get_Features

Report RTOS features.

```
unsigned long BIOS_Get_Features ( void );
```

## Parameters

-- none --

## Return Value

Bit field indicating features in this RTOS version with following bit assignments:

Bit 0: BGF_ETHERNET - Ethernet device for TCPIP
Bit 1: BGF_PPP_SERVER - PPP server
Bit 2: BGF_PPP_CLIENT - PPP client
Bit 3: BGF_WEB_SERVER - Web server
Bit 4: BGF_TELNET_SERVER - Telnet server
Bit 5: BGF_FTP_SERVER - FTP server
Bit 6: BGF_TFTP_SERVER - TFTP server
Bit 7: BGF_DHCP_CLIENT - DHCP client
Bit 8: BGF_IPV6 - IPv6 support
Bit 9: BGF_SSLWEB_SERVER - SSL Web server
....
Bit 16: BGF_I2C_BUS - I2C-Bus API
Bit 17: BGF_HW_API - Hardware API
Bit 18: BGF_RTOS_API - RTOS API
Bit 19: BGF_PACKET - Packet driver interface for ethernet
Bit 20: BGF_XMODEM - Serial XMODEM filetransfer
Bit 21: BGF_EXT_DISK - External disk interface
Bit 22: BGF_SOFT_SPI - Software SPI API
....
Bit 24: BGF_SNMP_MIB - SNMP MIB variables support (see **Get_SNMP_Data**)
Bit 25: BGF_UDP_CONFIG - UDP config server
Bit 26: BGF_PING_CLIENT - Ping client (see **Ping_Open**)
Bit 27: BGF_DEVICE_DRIVER - TCP/IP device driver
Bit 28: BGF_WEB_PUT - Webfile upload (Webserver PUT method)

The mask names stated here (e.g. BGF_PPP_SERVER) are defined in BIOS_API.H include file.

## Comments

The bit field returned is coded as so:

Bit=0: service or device is not available.
Bit=1: service or device is available.

## See Also

**BIOS_Get_RTOS_Version**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Power_Save

Power save.

```
void BIOS_Power_Save ( void );
```

## Parameters

-- none --

## Return Value

-- none --

## Comments

This function slows down the internal timer for the RTOS and puts the CPU in a halt mode until the next interrupt occurs. Call this function when your program is in idle state.

Power savings are marginal since DRAM is used by the CPU.

Note:
This function will affect the internal date/time.
Power consumption may differ slightly when the date code of the IPC@CHIP®is changed.

## See Also

BIOS_Power_SaveEx

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Power_SaveEx

Power save.

```
int BIOS_Power_SaveEx ( unsigned char mode );
```

## SC11/SC13 Parameters

*mode*

0x01: Restore full speed mode
0x02: Set 1/2 speed mode
0x04: Set 1/4 speed mode (serial baudrates > 57600 will no longer be accurate)
0x80: Enter sleep mode (RTOS 1ms timer and time/date will be affected)
0x81: Wakeup from sleep mode
0x20: Power down Ethernet controller (no Ethernet communication is possible)
0x21: Power up Ethernet controller

## SC1x3/SC23 Parameters

*mode*

0x01: Restore full speed mode
0x02: Set 1/2 speed mode
0x04: Set 1/4 speed mode
0x20: Power down Ethernet controller (no Ethernet communication is possible)
0x21: Power up Ethernet controller
0x22: Disable the Ethernet Energy Detect Power-Down mode
0x23: Enable the Ethernet Energy Detect Power-Down mode

## Return Value

0 if successful

## Comments

This function slows down the processor clock and puts the CPU in a powersave mode until the original mode is restored.

Note: Power consumption may differ slightly when the date code of the IPC@CHIP® is changed.

## SC12 Comments

For SC12 the `mode` parameter has no affect. The **BIOS_Power_Save** API is recommended for SC12.

## SC1x3/SC23 Comments

In mode 0x23 the Ethernet PHY is partially powered down. The PHY powers-up when there is valid energy from the line: 100Base-TX, 10 Base-T or auto negotiation.
If the user calls this function with AL==4 (Set 1/4 speed mode), the internal ethernet controller doesn't work in the 100 MBit mode. Because of this behaviour, it is recommended to set the **ethernet mode** at chip.ini fix to 10MBit Halfduplex.

The modes 0x02 and 0x04 influence the USB API. In mode 0x02 the USB API cannot be used at all. The SC1X3/SC23 will crash if you do. In mode 0x04 the USB API can only be used in Device mode, because Host mode needs better performance.

## See Also

**BIOS_Power_Save**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Install_Error_Handler

Install a user fatal error handler.

```
USER_ERROR_HANDLER BIOS_Install_Error_Handler ( USER_ERROR_HANDLER vector );
```

## Parameters

*vector*

Far pointer to callback procedure. Set to zero to un-install the handler.

## Return Value

Old handler (since SC1x RTOS V1.25 and SC1x3/SC23 RTOS V1.20)

## Comments

The user can install an error handler function, that will execute if a fatal error occurs in an application or inside of the @CHIP-RTOS.

The callback function must be be from:

```
void huge _pascal Err_Handler (int error_code)
```

The `error_code` input parameter can have the following values:

1: Invalid opcode (usually caused by corrupted memory), the calling task is suspended
2: Fatal kernel error (usually caused by corrupted memory or a task stack overflow)
3: Fatal internal TCPIP error, calling task is supended
4: TCPIP stack reaches memory limit
5: TCPIP memory allocation error
6: Ethernet bus error (hardware defect)
7: Ethernet link error(cable not connected?)
8: Flash write error ->Flash defect (IPC@CHIP®is no longer usable)
9: Low Memory error ->called if an alloc (system or user) failed

In all cases we recommend a reboot with BIOSint 0xA0 0x21 (see below). If the flash device has a defect, the IPC@CHIP® is no longer usable

**Important:**

Do not use any message printing inside your error handler if errorcode is 3 or 4 and telnet is part of your stdio. In this case your exit handler would hang inside of the print call.

## See Also

BIOS_Reboot

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.20 | V1.20 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Reboot

Reboot the CPU.

```
void BIOS_Reboot ( void );
```

## Parameters

-- none --

## Return Value

No return from this function

## Comments

This function works the same as the shell reboot command.

## See Also

BIOS_Install_Error_Handler

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## BIOS_Get_Device_Names

Get the IPC@CHIP® device names.

```
void BIOS_Get_Device_Names ( char far **fixed_name,
                             char far **configured_name );
```

## Parameters

*fixed_name*

Output parameter: Pointer to a string pointer which will receive a pointer to the fixed device name stored in the IPC@CHIP® flash.

*configured_name*

Output parameter: Pointer to a string pointer which will receive a pointer to the device name configured in CHIP.INI.

## Return Value

-- none --

## Comments

You cannot write to the string at `fixed_name` , because it is located in flash memory.

The string at `configured_name` is up to 20 characters long.

Both returned strings are zero terminated.

## See Also

**BIOS_Get_Device_NamesEx**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

**Device name** definition in CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Get_Device_NamesEx

Get the IPC@CHIP® device names. (Extended)

```
void BIOS_Get_Device_NamesEx ( char far **fixed_name,
                               char far **configured_name,
                               char far **beckproduct_name );
```

## Parameters

*fixed_name*

Output parameter: Pointer to a string pointer which will receive a pointer to the fixed device name stored in the IPC@CHIP® flash.

*configured_name*

Output parameter: Pointer to a string pointer which will receive a pointer to the device name configured in CHIP.INI.

*beckproduct_name*

Output parameter: Pointer to a string pointer which will receive a pointer to the fixed BECK product device name stored in the IPC@CHIP® flash.

## Return Value

-- none --

## Comments

You cannot write to the string at `fixed_name` , because it is located in flash memory.

The string at `configured_name` is up to 20 characters long.

All returned strings are zero terminated.

## See Also

**BIOS_Get_Device_Names**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Related Topics

**Device name** definition in CHIP.INI file

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Set_Memory_Gap

Set the memory gap used between the loaded DOS programs.

```
int BIOS_Set_Memory_Gap ( unsigned int paragraphs );
```

## Parameters

*paragraphs*

Size of memory gap in 16 byte paragraphs: range [0 .. 2048]

## Return Value

0: Successful
-1: Failure, invalid value in `paragraphs`

## Comments

This function sets the size of the memory gap that will be used between subsequently loaded DOS programs. This gap serves as a memory reserve.

Some programs compiled with Borland C 5.02 try to increase the size of their program memory block at runtime before, for example, opening a file with Borland C-Library function fopen. (The programs calls int21h **0x4A**. This happens down inside the C-Library fopen function and is not directly visible to the application programmer.)

This memory resize call fails if another program has been loaded after the one making the call due to now there is no memory space left for increasing the memory size of the first program. The program returns from fopen with an error. The global program variable errno is set to value 8 (not enough memory). To prevent this error the size of a memory gap between two loaded programs can be controlled. The value is defined in terms of memory paragraphs (1 paragraph = 16 bytes). This value can also defined in the **CHIP.INI** file.

This strategy could fail in the cases where programs are terminated and are started again.

It is usually not necessary to set this entry, if the application does not show the described error. Only if a C-Library function call sets errno to 8, should this value be defined. We recommend in case of this error, a memory gap value of 128 paragraphs (2048 Bytes).

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## BIOS_Write_Persistent_Data

Write Persistent User Data.

```
void BIOS_Write_Persistent_Data ( const void far *data,
                                  unsigned char size );
```

## Parameters

*data*

data buffer which contains data to write

*size*

number of bytes to write, max. 192

## Return Value

none

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## BIOS_Read_Persistent_Data

Read Persistent User Data.

```
void BIOS_Read_Persistent_Data ( void far *data,
                                 unsigned char size );
```

## Parameters

*data*

data buffer where read data should be stored

*size*

number of bytes to read, max. 192

## Return Value

none

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document




# @CHIP-RTOS C Library - CAN API

Header File: `#include <CAN_API.H>`

## Configuration Functions

| | |
|---|---|
| CAN_Open_Port | Open specified CAN port |
| CAN_Rx_Filters | Configure receive filters |
| CAN_Reconfig | Reconfigure CAN port |
| CAN_Control | Enable, disable and purge |
| CAN_Close_Port | Shut down specified CAN port |

## Transmit Functions

| | |
|---|---|
| CAN_Send | Output a CAN message |

## Receive Functions

| | |
|---|---|
| CAN_Recv | Pickup a received CAN message |
| CAN_Peek | Look at received CAN message, leave in queue |

## Status Functions

| | |
|---|---|
| CAN_Status | Check port status |
| CAN_Error_Counts | Check port error counts |
| CAN_Event_Sleep | Sleep until port event occurs |
| CAN_Set_Callback | Install/Remove ISR callback |

List of C Libraries
@CHIP-RTOS Main Index

End of document




# @CHIP-RTOS C Library - Common Gateway Interface API

Header File: `#include <CGIAPI.H>`

## CGI Functions

| | |
|---|---|
| CGI_Install | Install a CGI function |
| CGI_Delete | Remove a CGI function |
| CGI_SetMainPage | Select a new main page |
| CGI_GetMainPage | Get main page name |
| CGI_SetMainPagePost | Select a new main page (POST method) |
| CGI_GetMainPagePost | Get main page name (POST method) |
| CGI_SetRootDrive | Set Web server's root drive |
| CGI_GetRootDrive | Get Web server's root drive |
| CGI_SetRootDir | Set Web server's root directory |
| CGI_GetRootDir | Get Web server's root directory |
| CGI_GetArgument | Returns the next available argument |
| CGI_UrlEncode | Encodes a URL string |
| CGI_UrlDecode | Decodes a URL string |
| CGI_GetCookie | Returns the next available cookie |
| CGI_SetCookie | Creates a new cookie or modifies an old one |
| CGI_GetFormItem | **Deprecated!** Split a form item into name and value |
| CGI_GetNextItem | **Deprecated!** Return pointer to the next form item |
| CGI_Install_Func_OnErr | Install error cgi function |
| CGI_Install_MIME_Table | Install a custom MIME table |
| CGI_Remove_MIME_Table | Remove the custom MIME table |
| CGI_Install_ContentEncoding_Table | Install the custom content-encoding table |
| CGI_Remove_ContentEncoding_Table | Remove the custom content-encoding table |

## SSI Functions

| | |
|---|---|
| **SSI_Set_File_Filter** | Set a SSI file filter |
| **SSI_Register_Tag** | Register a SSI tag |
| **SSI_Deregister_Tag** | Deregister a SSI tag |
| **SSI_Set_TagLength** | Set the maximum SSI tag length |
| **SSI_Send_Buffer** | Send a buffer over a HTTP connection |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CGI_Install

Install a CGI function.

```
int CGI_Install ( CGI_Entry far *cgiptr );
```

## Parameters

*cgiptr*

Pointer to a temporary `CGI_Entry` **type** structure.

## Return Value

0: success
non-zero: Failure, return value 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 `cgiptr` need not be persistent.

## See Also

**CGI_Delete**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

`cgistat` **command** line

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Common Gateway Interface API

## CGI_Delete

Remove a CGI function.

```
int CGI_Delete ( char far * PathPtr );
```

## Parameters

*PathPtr*

Pointer to the null terminated URL path name.

## Return Value

0: success
non-zero: Failure, return value 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.

## See Also

CGI_Install  CGI_SetMainPage

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Related Topics

`cgistat` command line

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CGI_SetMainPage

Set a new main page.

```
int CGI_SetMainPage ( const char far *mainPageName );
```

## Parameters

*mainPageName*

Pointer to name of new main page.

## Return Value

0: success
non-zero: Failure, return value contains error code CGI_INVALID_NAME

## Comments

The string at mainPageName is null terminated with a maximum length of 64 characters (not counting the terminating zero).

## See Also

CGI_GetMainPage  CGI_SetRootDir

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

End of document

## CGI_GetMainPage

Get main page name.

```
int CGI_GetMainPage ( char far **mainPageName );
```

## Parameters

*mainPageName*

Contains pointer to current main page name.

## Return Value

0: The variable mainPageName contains a pointer to a string containing name of current main page

## Comments

The referenced main page name string is null terminated and is in the RTOS's data space.

The caller must treat this string as a constant and must not modify it.

## See Also

CGI_SetMainPage CGI_GetRootDir

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

End of document

## CGI_SetMainPagePost

Set a new main page (POST method).

```
int CGI_SetMainPagePost ( const char far *mainPageName );
```

## Parameters

*mainPageName*

Pointer to name of new main page.

## Return Value

0: success
non-zero: Failure, return value contains error code CGI_INVALID_NAME

## Comments

The string at mainPageName is null terminated with a maximum length of 64 characters (not counting the terminating zero).

## See Also

CGI_GetMainPagePost  CGI_SetRootDir

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.20 | V1.20 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.14 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CGI_GetMainPagePost

Get main page name (POST method).

```
int CGI_GetMainPagePost ( char far **mainPageName );
```

## Parameters

*mainPageName*

Contains pointer to current main page name.

## Return Value

0: The variable mainPageName contains a pointer to a string containing name of current main page

## Comments

The referenced main page name string is null terminated and is in the RTOS's data space.

The caller must treat this string as a constant and must not modify it.

## See Also

CGI_SetMainPagePost  CGI_GetRootDir

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.25 | V1.25 | V1.25 | V1.20 | V1.20 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.14 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CGI_SetRootDrive

Set Web server's root drive.

```
int CGI_SetRootDrive ( const char driveLetter );
```

## Parameters

*driveLetter*

Letter of the new drive.

## Return Value

0: success
non-zero: Failure, return value contains error code CGI_INVALID_DIR

## Comments

The driveLetter one in-case-sensetive character which specifies the new web server root drive (for example 'A', 'B', 'b', ...).

## See Also

CGI_GetRootDrive  CGI_GetRootDir  CGI_SetRootDir

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.26 | V1.26 | V1.26 | V1.21 | V1.21 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.15 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Common Gateway Interface API

## CGI_GetRootDrive

Get Web server's root drive.

```
char CGI_GetRootDrive ( );
```

## Return Value

Letter which specifies the web servers root drive.

## Comments

The return value specifies the new web server root drive (for example 'A', 'B', 'b', ...).

## See Also

CGI_SetRootDrive  CGI_SetRootDir  CGI_GetRootDir

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.26 | V1.26 | V1.26 | V1.21 | V1.21 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.15 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

# @CHIP-RTOS C Library - Common Gateway Interface API

---

## CGI_SetRootDir

Set Web server's root directory.

```
int CGI_SetRootDir ( const char far *rootDirName );
```

## Parameters

*rootDirName*

Pointer to the name of new root directory for Web server.

## Return Value

0: success
non-zero: Failure, return value contains error code CGI_INVALID_DIR

## Comments

The string at `rootDirName` is null terminated with a maximum length of 64 characters (not counting the terminating zero).

## See Also

CGI_GetRootDir  CGI_GetRootDrive  CGI_SetMainPage

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries

End of document

# @CHIP-RTOS C Library - Common Gateway Interface API

## CGI_GetRootDir

Get Web server's root directory.

```
int CGI_GetRootDir ( char far **rootDirName );
```

## Parameters

*rootDirName*

Contains pointer to root directory name.

## Return Value

0: The variable rootDirName contains a pointer to the Web server's root directory name.

## Comments

The referenced root directory name string is null terminated and is in the RTOS's data space.

The caller must treat this string as a constant and must not modify it.

## See Also

CGI_SetRootDir CGI_SetRootDrive CGI_GetMainPage

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries

End of document

## CGI_GetArgument

Returns the next available argument (name-value pair).

```
int CGI_GetArgument ( char far **name,
                      char far **value,
                      rpCgiPtr CgiRequest);
```

## Parameters

*name*

Output Parameter: A pointer to a valid char-pointer variable. After the function call the char-pointer variable points to a c-string with the argument-name.

*value*

Output Parameter: A pointer to a valid char-pointer variable. After the function call the char-pointer variable points to a c-string with the argument-value.

*CgiRequest*

Input Parameter: The cgi request pointer which was passed to the cgi function.

## Return Value

CGI_ARGUMENT_ERR_OK: success
CGI_ARGUMENT_ERR_NONE: there was no more argument found

## Comments

On the first call the function returns the first found argument. On next calls the function returns the next found argument, until no more arguments could be found.
Note that this function must be called within the context of a cgi function. The pointers returned by the function are valid only while the cgi function is active. On return from the cgi function the pointer will become invalid.
The name, value pairs returned by this function are already decoded. You don't have to call **CGI_UrlDeocde** yourself.

Example:

```
char *name;
char *value;

while ( CGI_GetArgument( &name, &value, CgiRequest ) == CGI_ARGUMENT_ERR_OK )
{
   printf("Argument found:\r\n"
          "Name : %s\r\n"
          "Value: %s\r\n",
          name, value);
}
```

## See Also

CGI_GetFormItem

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.10 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

# @CHIP-RTOS C Library - Common Gateway Interface API

## CGI_UrlEncode

Encodes a URL string

```
void CGI_UrlEncode ( char *dst,
                     const char far *src );
```

## Parameters

*dst*

Output Parameter: Pointer to the buffer the encoded string is to be stored in; must be large enough. The encoded string can be up to three times as large as the original string.

*src*

String that is to be encoded

## Comments

In URL-encoded strings all non-alphanumerical characters except '-', '_' and '.' are encoded as a hexadecimal value preceded by a percent sign (e.g. %26 for the ampersand). The space character is replaced by a plus sign. The text "Test 123 .&%" (without the quotes) would be encoded as "Test+123+.%26%25".
This kind of encoding is for example used to pass arguments to a CGI page with the URL.

## See Also

CGI_UrlDecode

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.12 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Common Gateway Interface API

## CGI_UrlDecode

Decodes a URL string

```
void CGI_UrlDecode ( char far *dst,
                     const char far *src );
```

## Parameters

*dst*

Output Parameter: Pointer to the buffer the decoded string is to be stored in; must be large enough.

*src*

URL-encoded string that is to be decoded

## Comments

In URL-encoded strings all non-alphanumerical characters except '-', '_' and '.' are encoded as a hexadecimal value preceded by a percent sign (e.g. %26 for the ampersand). The space character is replaced by a plus sign. The text "Test 123 .&%" (without the quotes) would be encoded as "Test+123+.%26%25".
This kind of encoding is for example used to pass arguments to a CGI page with the URL.

You can pass the same pointer as `src` and `dst` parameter. The encoded string will in this case be overwritten with the decoded string.
Example:
`CGI_UrlDecode(url, url);`

## See Also

CGI_UrlEncode
CGI_GetArgument

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.12 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CGI_GetCookie

Returns the next available cookie.

```
int CGI_GetCookie ( char far **name,
                    char far **value,
                    char far **domain,
                    char far **path,
                    char far **port,
                    char far **version,
                    rpCgiPtr CgiRequest);
```

## Parameters

*name*

Output Parameter: A pointer to a valid char-pointer variable. After the function call the char-pointer variable points to a c-string with the cookie-name.

*value*

Output Parameter: A pointer to a valid char-pointer variable. After the function call the char-pointer variable points to a c-string with the cookie-value.

*domain*

Output Parameter: A pointer to a valid char-pointer variable. After the function call the char-pointer variable points to a c-string with the cookie-domain. If you are not interest in the cookie-domain then you can pass a null pointer.

*path*

Output Parameter: A pointer to a valid char-pointer variable. After the function call the char-pointer variable points to a c-string with the cookie-path. If you are not interest in the cookie-path then you can pass a null pointer.

*port*

Output Parameter: A pointer to a valid char-pointer variable. After the function call the char-pointer variable points to a c-string with the cookie-port. If you are not interest in the cookie-port then you can pass a null pointer.

*version*

Output Parameter: A pointer to a valid char-pointer variable. After the function call the char-pointer variable points to a c-string with the cookie-version. If you are not interest in the cookie-version then you can pass a null pointer.

*CgiRequest*

Input Parameter: The cgi request pointer which was passed to the cgi function.

## Return Value

CGI_COOKIE_ERR_OK: success
CGI_COOKIE_ERR_NONE: there was no more cookie found

## Comments

On the first call the function returns the first found cookie. On next calls the function returns the next found cookie, until no more cookie could be found.
Note that this function must be called within the context of a cgi function. The pointers returned by the function are valid only while the cgi function is active. On return from the cgi function the pointer will become invalid.

Example:

```
char *name;
char *value;
char *domain;
char *path;
char *port;
char *version;

while ( CGI_GetCookie( &name, &value, &domain, &path, &port, &version ) == CGI_COOKIE_ERR_OK)
{
    // do something with the found cookie
    printf( "Cookie found:\r\n"
            "Name   : %s\r\n",
            "Value  : %s\r\n",
            "Domain : %s\r\n",
            "Path   : %s\r\n",
            "Port   : %s\r\n",
            "Version: %s\r\n",
            name, value, domain, path, port, version );
}
```

## See Also

CGI_SetCookie

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.10 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## CGI_SetCookie

Creates a new cookie or modifies an old one.

```
int CGI_SetCookie ( const char far *name,
                    const char far *value,
                    const char far *domain,
                    const unsigned long far *expires,
                    const char far *path,
                    const int secure,
                    rpCgiPtr CgiRequest);
```

## Parameters

*name*

Input Parameter: A pointer to a c-string which represents cookies-name.

*value*

Input Parameter: A pointer to a c-string which represents cookies-value.

*domain*

Input Parameter: A pointer to a c-string which represents cookies-domain. If you are not interest in the cookie-domain then you can pass a null pointer.

*expires*

Input Parameter: A pointer to an unsigned long which represents cookies-expire-time. If you are not interest in the cookie-expire-time then you can pass a null pointer.

*path*

Input Parameter: A pointer to a c-string which represents cookies-path. If you are not interest in the cookie-path then you can pass a null pointer.

*secure*

Input Parameter: This ineger value defines the secure state of the cookie.
0 : no secure cookie
!=0 : secure cookie

*CgiRequest*

Input Parameter: The cgi request pointer which was passed to the cgi function.

## Return Value

CGI_COOKIE_ERR_OK: Success, the cookie was created
CGI_COOKIE_ERR_SIZE: The cookie parameters in sum are too large. All cookies together must not be larger than 1024 bytes.
CGI_COOKIE_ERR_PARAM: Invalid parameters, e.g. no cookie-name .

## Comments

The argument strings are copied internal, so they are not required after the function call and could by deleted or modified.
Note that the date and time and also the timezone have to be set when you are using the `expires` argument.
This function must be called within the context of a cgi function. The pointers returned by the function are valid only while the cgi function is active. On return from the cgi function the pointer will become invalid.

Example:

```
int result = CGI_SetCookie("username",
                           "otto",
                           NULL,
                           NULL,
                           NULL,
                           0,
                           CgiRequest);
if (result != CGI_COOKIE_ERR_OK)
{
   // ERROR!
}
```

## See Also

CGI_GetCookie

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.15 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.10 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS C Library - Common Gateway Interface API

---

## CGI_GetFormItem

Split a form item into name and value.

```
void CGI_GetFormItem ( const char far *ArgumentBufferPtr,
                       FormItem far *formitem );
```

This function is **deprecated**! Use **CGI_GetArgument** instead.

## Parameters

*ArgumentBufferPtr*

Input Parameter: Pointer to argument buffer to be parsed.

*formitem*

Input/Output Parameters: Pointer to a `FormItem` **type** structure

## Return Value

-- none --

## Comments

On initial call, the `ArgumentBufferPtr` provided by the caller is a copy of the `fArgumentBufferPtr` member of the `rbCgi` **structure** passed by the Web server to the CGI callback function. On subsequent calls to pick up additional form items, the pointer returned from the **CGI_GetNextItem** function can be used here.

Prior to calling the two members of the `FormItem` **structure** must be set by the caller. Both member pointers in the `FormItem` data structure reference buffers allocated by the user, which will receive strings produced by this API call.

## See Also

**CGI_GetNextItem** **GGI_GetArgument**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS - Deprecated functions



## Deprecated

In software development the term **deprecated** is used to discourage usage of a particular software feature. Usually this is due to that the feature has been superseded by a new/better version.

The IPC@CHIP® documentation marks functions as **deprecated** which should no longer be used by the software developer since there is a new/better function available. The function which should be used instead is listed in the respective **deprecated** notice. Nevertheless the various **deprecated** functions will continue to be supported for backwards compatibility with older software.

End of document

## CGI_GetNextItem

Get the next form item.

```
int CGI_GetNextItem ( char far *ArgumentBufferPtr,
                      char far **nextItemString );
```

This function is **deprecated**! Use **CGI_GetArgument** instead.

### Parameters

*ArgumentBufferPtr*

CGI request argument pointer.

*nextItemString*

Pointer to the found item.

### Return Value

0: Item was found
-1: No more items found

### Comments

Most forms have more than one item. This function searches for the next form item in a CGI request argument string. It can only be used after a **CGI_GetFormItem** call.

The string referenced by `ArgumentBufferPtr` is not modified in any way by this function.

### See Also

**CGI_GetFormItem** **GGI_GetArgument**

### RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CGI_Install_Func_OnErr

Install CGI function on http errors

```
int CGI_Install_Func_OnErr( unsigned int error_state,
                            void far *CgiFuncOnErr);
```

## Parameters

*error_state*

http error state (see below)

*CgiFuncOnErr*

CGI Function pointer

## Return Value

0: CGI function installed
-1: CGI function not installed

## Comments

The CGI function must be implemented in the same way than a CGI function, installed with **CGI_INSTALL**. For deleting an entry, install a null pointer.

Possible Http error states(parameter error_state):
07: 401 Unauthorized
08: 404 not found
10: 500 Server error (File system error)
22: 503 Service not available (Out of memory)

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V1.12 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Common Gateway Interface API

## CGI_Install_MIME_Table

Install user custom MIME table

```
int CGI_Install_MIME_Table( unsigned int entries,
                            CUSTOM_MIMETYPES_TABLE far *mimeTable);
```

## Parameters

*entries*

Number of entries in MIME table

*mimeTable*

Pointer to custom **MIME table**

## Return Value

0: OK
-1: Error

## Comments

The MIME table must be persistent. The RTOS does not make a copy.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|

| V2.04 |
|---|

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## CGI_Remove_MIME_Table

Deinstall user custom MIME table

```
int CGI_Remove_MIME_Table( void );
```

## Return Value

0: OK
-1: Error

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## CGI_Install_ContentEncoding_Table

Install user custom content-encoding table

```
int CGI_Install_ContentEncoding_Table( unsigned int entries,
            CUSTOM_CONTENTENCODING_TABLE far *contentEncodingTable);
```

## Parameters

*entries*

Number of entries in content-encoding table (must be equal to the user custom mime table size).

*contentEncodingTable*

Pointer to custom **content-encoding table**

## Return Value

0: OK
-1: Error (wrong number of entries)

## Comments

The content encoding table references to the custom MIME table. So the content encoding table assigns every mime type defined in the MIME table a correspondent content encoding type.
The MIME table must be persistent. The RTOS does not make a copy.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.16 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.11 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Common Gateway Interface API

## CGI_Remove_ContentEncoding_Table

Deinstall user custom content-encoding table

```
int CGI_Remove_ContentEncoding_Table( void );
```

## Return Value

0: OK
-1: Error

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.16 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.11 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## SSI_Set_File_Filter

Set Server Side Include (SSI) file filter

```
int SSI_Set_File_Filter( char far * matchStr,
                         int matchType,
                         int server);
```

## Parameters

*matchStr*

Pointer to null-terminated string depending on the machted type

*matchType*

Match type, possible values are:

`SSI_MATCH_FILE_NAME(0x01)` = File name, Example: "table.htm"
`SSI_MATCH_FILE_EXT (0x02)` = File extension, Example: "htm"
`SSI_MATCH_FILE_PATH(0x03)` = Full path to file, Example: "/web/ssi/table.htm"
`SSI_MATCH_DIR_NAME (0x04)` = Directory name, Example: "ssi"
`SSI_MATCH_DIR_PATH (0x05)` = Directory path, Example: "/web/ssi"

*server*

0 = register for HTTP server
1 = register for HTTPS (SSL) server

## Return Value

0: success
non-zero: Failure, return value contains error code CGI_INVALID_METHOD

## Comments

SSI commands are special HTML comment tags in web pages, and parsed by the web server while the pages are being served. They allow dynamically generated content to be inserted into a static page, without having to dynamically generate the whole page.
There's no official standard for SSI command. Thus we provide an API to register a SSI tag string and a handler function. When a registered SSI tag is found in a page being served, the handler function will be called to process it.

## See Also

SSI_Register_Tag  SSI_Set_TagLength

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSI_Register_Tag

Register Server Side Include (SSI) tag string

```
int SSI_Register_Tag( const char far * tagStr,
                      SSI_Handler handler,
                      int server);
```

## Parameters

*tagStr*

Tag string, Pointer to null-terminated string

*handler*

Pointer to callback function

**Callback function prototype:**

```
typedef
  int huge (far *SSI_Handler) (unsigned long conhandle, const char far *
tagPtr);
```

**Possible callback function return values:**
`SSI_OUTPUT_ENABLE(0x00)` = Enable web server to output static content
`SSI_OUTPUT_DISABLE(0x01)` = Disable web server to output static content
`SSI_ABORT_PAGE(0x02)` = Stop processing the current file
other = The web server will send an error message and stop serving the HTTP request

*server*

0 = register for HTTP server
1 = register for HTTPS (SSL) server

## Return Value

0: success
non-zero: Failure, return value contains error code CGI_INVALID_METHOD or CGI_NO_FREE_ENTRY

## Comments

Currently only one tag per server can be registered.

## See Also

SSI_Set_File_Filter  SSI_Set_TagLength  SSI_Send_Buffer

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSI_Deregister_Tag

Deregister Server Side Include (SSI) tag string

```
int SSI_Deregister_Tag( SSI_Handler handler,
                        int server);
```

## Parameters

*handler*

Pointer to installed callback function

*server*

0 = deregister for HTTP server
1 = deregister for HTTPS (SSL) server

## Return Value

0: success
non-zero: Failure, return value contains error code CGI_INVALID_METHOD

## See Also

SSI_Set_File_Filter  SSI_Set_TagLength  SSI_Register_Tag

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.21 | V1.21 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.15 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSI_Set_TagLength

Set maximum length of Server Side Include (SSI) tag string

```
int SSI_Set_TagLength( int tagLength );
```

## Parameters

*tagLength*

length in bytes (default is 90)

## Return Value

0: success
non-zero: Failure, return value contains error code CGI_INVALID_METHOD

## Comments

Smaller values will optimize the performance of the SSI parser.

## See Also

SSI_Set_File_Filter  SSI_Set_TagLength

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|

| V2.02 |
| --- |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSI_Send_Buffer

Transmit a user buffer to the HTTP client through the HTTP connection

```
int SSI_Send_Buffer( unsigned long conHandle,
                     const char far *buffer,
                     int bufferlen,
                     int flag );
```

## Parameters

*conHandle*

connection handle, given by callback function

*buffer*

pointer to send buffer

*bufferlen*

length of buffer in bytes, max. 32767

*flag*

flag, set to 1

## Return Value

0: success
non-zero: Failure, return value contains error code

## Comments

Smaller values will optimize the performance of the SSI parser.

## See Also

SSI_Set_File_Filter  SSI_Register_Tag

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS C Library - External Disk API

---

Header File: `#include <EXTDISK.H>`

## Driver Functions

| | |
|---|---|
| Extdisk_install_drive | Install a drive B: |
| Extdisk_remove_drive | Deinstall the drive B: |
| Extdisk_install_drive2 | Install a drive D: |
| Extdisk_remove_drive2 | Deinstall the drive D: |

## Status Functions

| | |
|---|---|
| Extdisk_install_callback | Install a notification callback |
| Extdisk_remove_callback | Deinstall the notification callback |

List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---

# @CHIP-RTOS C Library - External Disk API

## Extdisk_install_drive

Install a drive B:

```
int Extdisk_install_drive ( unsigned long totalsectors );
```

## Parameters

*totalsectors*

Total sector count of disk

## Return Value

= 0: success
!= 0: failed

## Comments

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    Sector number (BX is MSH of unsigned long)
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.

## See Also

Extdisk_remove_drive

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Extdisk_remove_drive

Deinstall the drive B:

```
int Extdisk_remove_drive ( void );
```

## Parameters

-- none --

## Return Value

= 0: success
!= 0: failed

## See Also

Extdisk_install_drive

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## Extdisk_install_drive2

Install a drive D:

```
int Extdisk_install_drive2 ( unsigned long totalsectors );
```

## Parameters

*totalsectors*

Total sector count of disk

## Return Value

= 0: success
!= 0: failed

## Comments

Prior to this call the application must provide a disk read/write function installed at interrupt 0xB2. This installed handler function should expect:

```
AX       1 for write, 0 for read.
BX,DX    Sector number (BX is MSH of unsigned long)
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.

## See Also

Extdisk_remove_drive2

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Extdisk_remove_drive2

Deinstall the drive D:

```
int Extdisk_remove_drive2 ( void );
```

## Parameters

-- none --

## Return Value

= 0: success
!= 0: failed

## See Also

Extdisk_install_drive2

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.06 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## Extdisk_install_callback

Install a notification callback function

```
int Extdisk_install_callback ( EXTDISK_CALLBACK callback );
```

## Parameters

*callback*

Pointer to the callback function of the type EXTDISK_CALLBACK

EXTDISK_CALLBACK is defined as follows:

```
typedef void (far * EXTDISK_CALLBACK)(unsigned int event);
```

## Return Value

= 0: success
!= 0: failed

## Comments

The external drives B: and D: may be hot-plugable. This means that the drive medium might be removed or inserted while your application is running. In some cases it may be important for you to be notified if a medium is removed or inserted, for example if one of your tasks has set its working directory to the external drive. By default @CHIP-RTOS will reset the working directory of all tasks that previously had their working directory on a drive that has been removed to A:\. However this might not be appropriate. So a reaction on the removal of a hot-pluggable medium might be to set a new valid working directory. To be notified of insertion and removal events, install an EXTDISK callback function.

A maximum of 12 EXTDISK callback functions can be installed.

The following events may be passed along with the callback function's event parameter:
EXTDISK_EVENT_INSTALL_B = 0x01
Drive B: has been installed

EXTDISK_EVENT_REMOVE_B = 0x02
Drive B: has been uninstalled

EXTDISK_EVENT_INSTALL_D = 0x03
Drive D: has been installed

EXTDISK_EVENT_REMOVE_D = 0x04
Drive D: has been uninstalled

## Example

The following example demonstrates the general usage of the notification callback function:

```
void huge callback(unsigned int event)
{
  switch(event)
  {
    case EXTDISK_EVENT_INSTALL_B:
      ...  // Event handling
      break;

    case ...
  }
}

void main()
{
  if(Extdisk_install_callback(callback) != 0)
    ...  // Error handling

  ...

  if(Extdisk_remove_callback(callback) != 0)
    ...  // Error handling
}
```

## See Also

**Extdisk_remove_callback**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Extdisk_remove_callback

Uninstall the notification callback function

```
int Extdisk_remove_callback ( EXTDISK_CALLBACK callback );
```

## Parameters

*callback*

Pointer to the callback function of the type EXTDISK_CALLBACK (see **Extdisk_install_callback** for details)

## Return Value

= 0: success
!= 0: failed

## See Also

**Extdisk_install_callback**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

End of document




# @CHIP-RTOS C Library - Fossil API

---

Header File: `#include <FOSSIL.H>`

Here is the API definition for access to the serial ports.

The @Chip-RTOS offers the Fossil API for serial port communication. The Fossil standard uses software interrupt 0x14. These functions provide access to the @Chip-RTOS internal serial port driver for receiving and sending data via the serial ports.

Here is a short description how the @Chip-RTOS internal serial port driver operates:

Each serial port has two software buffers (queues), one for data pending transmission and one for storing received data bytes. The default size of each queue is 1024 bytes. The size of these transmit and receive queues are configurable via
**CHIP.INI**
entries.

Transfers between these software queues and the serial port hardware are carried out by either Direct Memory Access (DMA) hardware or by hardware interrupt (IRQ) driven software. By default, the available DMA machines are applied to the serial port receivers. This configuration reduces the likelihood of character loss at the receiver. This leaves the serial port transmitters operating with hardware interrupt (IRQ) driven software. Alternately, receivers can be configured for interrupt driven mode (IRQ receive mode) with the
**CHIP.INI**
options, which frees up the DMA device for other usage. The DMA can be applied to transmitters with the SERIAL
**SEND_DMA**
option.

The serial port hardware issues a signal to either software (IRQ hardware interrupt) or to the DMA when ever the serial port transmit register is empty or a receiver byte is ready. This signal initiates the next byte transfer between the appropriate software queue and the hardware register, in or out. (This discussion is slightly over simplified. The driver actually uses an additional intermediate RAM buffer for DMA transfers.)

Note that the serial port hardware send/receive buffers are only one byte (four bytes) deep on the SC1x (SC1x3/SC23). So interrupt driven receivers (as opposed to DMA driven) can easily lose characters, particularly at higher baud rates.

The serial ports may also be referred to as a UART (abbreviation for "Universal Asynchronous Receiver/Transmitter").

For some useful comments see **Programming notes**

## Configuration Functions

| | |
|---|---|
| fossil_init | Initialize the fossil driver |
| fossil_deinit | Deinitialize fossil driver |
| fossil_setbaud | Set baud rate |
| fossil_setbaud_std | Set baud rate to standard rates |
| fossil_set_extctrl | **Deprecated!** Extended line control initialization |
| fossil_set_flowcontrol | Enable/disable flow control |
| fossil_force_rts | Force RTS signal |
| fossil_read_cts | Read CTS signal |
| fossil_user_callback | Install a Fossil User Callback Function |
| -- Status Functions -- | |
| fossil_status_request | Get port status |
| fossil_get_driver_info | Get driver info |
| fossil_get_bytes_in_uart | Report number of bytes in UART transmit hardware buffers. |
| -- RS-485 Control -- | |
| fossil_set_rs485 | Enable/Disable RS485 mode |
| fossil_set_rs485_txenable | Select RS485 pin |

## Transmitter Functions

| | |
|---|---|
| fossil_putbyte_wait | Put byte in output buffer, wait if needed |
| fossil_putbyte | Put byte in output buffer (without wait) |
| fossil_writeblock | Write a block of data |
| fossil_flush_output | Await empty transmit buffer |
| fossil_purge_output | Purge output buffer |
| fossil_send_break | Send break |
| fossil_enable_transmitter | Enable/Disable transmitter |

## Receiver Functions

| | |
|---|---|
| fossil_getbyte_wait | Get byte from input buffer (with wait) |
| fossil_getbyte | Get byte from input buffer (no wait) |
| fossil_peek_input | Peek if next byte is available |
| fossil_readblock | Read block of data |
| fossil_purge_input | Purge input buffer |
| fossil_enable_receiver | Enable/Disable receiver |

## External Port Functions

| | |
|---|---|
| **fossil_register_external_port** | Register an external port |
| **fossil_unregister_external_port** | Unregister an external port |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_init

Initialize the fossil driver for specified port

```
int fossil_init ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

0x1954, signature from Fossil driver

## Comments

Use this function to initialize the fossil interface for a specified 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.

## See Also

**fossil_deinit** **fossil_setbaud**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_deinit

Deinitialize fossil driver.

```
void fossil_deinit ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

None

## Comments

This function restores the specified RS-232 port to its default settings.

## See Also

**fossil_init** **fossil_setbaud**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_setbaud

Set baud rate and port configuration.

```
int fossil_setbaud (int port, long baudrate, int parity,
                    int wordlen, int stopbits );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*baudrate*

Desired baud rate, bits per second

*parity*

Enumeration type:
FOSSIL_PARITY_NO = 0
FOSSIL_PARITY_ODD = 1
FOSSIL_PARITY_EVEN = 2
FOSSIL_PARITY_MARK = 3
FOSSIL_PARITY_SPACE = 4

*wordlen*

Bits per character (7 or 8)

*stopbits*

Stop bits (1 or 2)

## Return Value

-1 = Illegal parameter
Else status code (bit field):

bit 6: FOSSIL_OUTPUT_BUFFER_EMPTY
bit 5: FOSSIL_OUTPUT_NOT_FULL
bit 4: FOSSIL_LINE_BREAK Line break detected
bit 3: FOSSIL_FRAMING_ERROR Framing error detected
bit 2: FOSSIL_PARITY_ERROR Parity error detected
bit 1: FOSSIL_OVERRUN_ERROR Set if overrun occurred on receiver.
bit 0: FOSSIL_DATA_AVAILABLE Set if data is available in receiver buffer.

The header file contains the indicated mask definitions.

## Comments

Set baud rate and other parameters for specified port.

For standard baud rates, this function uses Fossil interrupt 0x14 Service **0x00**.

For non-standard baud rates the HAL interrupt 0xA1 "Get Frequencies" function **0x8A** is used to determine the maximum baud rate. Based on this maximum baud rate a baud rate divider is computed for the "Extended Line Control Initialization" Fossil interrupt **0x81** service. Then the Fossil interrupt **0x03** service reports the port's status.

If the DMA mode (send or receive mode) is enabled at **CHIP.INI**, the following port settings are not allowed:

1. 8N2 (this limitation applies only to SC1x systems)
2. 8S1
3. 8M1

## SC1x Comments

Two stop bits are only available if no parity is set.

## SC1x3/SC23 Comments

Two stop bits are only available if no parity is set or word length is equal to 7 bits.

## See Also

**fossil_deinit** **fossil_set_extctrl** **fossil_set_flowcontrol**
**fossil_setbaud_std**

## RTOS API

This library function may invoke following RTOS software interrupts, depending on which are needed for the specified baudrate:

**HAL_GET_FREQUENCIES**
Fossil service **0x00**
Fossil service **0x81**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_setbaud_std

Set a standard baud rate and port configuration.

```
int fossil_setbaud_std( int port_idx,
                        unsigned char config ) ;
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:

`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*config*

Configuration parameter:
This bitmap field specifies the communication parameters.

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 |

Bits 1-0: Word length

| | |
|---|---|
| 10 | 7 bits |
| 11 | 8 bits |

## Return Value

Undefined if `port_idx` parameter is invalid
Else status code (bit field):
bit 6: FOSSIL_OUTPUT_BUFFER_EMPTY
bit 5: FOSSIL_OUTPUT_NOT_FULL
bit 4: FOSSIL_LINE_BREAK Line break detected
bit 3: FOSSIL_FRAMING_ERROR Framing error detected
bit 2: FOSSIL_PARITY_ERROR Parity error detected
bit 1: FOSSIL_OVERRUN_ERROR Set if overrun occurred on receiver.
bit 0: FOSSIL_DATA_AVAILABLE Set if data is available in receiver buffer.

## Comments

This API provides an interface to the Fossil interrupt 0x14 Service **0x00** requiring less code than used by the alternative **fossil_setbaud** API. This API is restricted to the standard UART baud rates listed above.

For applications which operate with only a few fixed UART configurations, using this API with a constant defined for the `config` argument will save some code space.

If the DMA mode (send or receive mode) is enabled at **CHIP.INI**, the following port settings are not allowed:

1. 8N2 (this limitation applies only to SC1x systems)
2. 8S1
3. 8M1

## SC1x Comments

Two stop bits are only available if no parity is set.

## SC1x3/SC23 Comments

Two stop bits are only available if no parity is set or word length is equal to 7 bits.

## See Also

**fossil_deinit** **fossil_set_extctrl** **fossil_set_flowcontrol**
**fossil_setbaud**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|

| V2.08 |
| --- |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_set_extctrl

Extended line control initialization.

```
void fossil_set_extctrl ( int port,
                          int baud_divider,
                          FOSSIL_PARITY parity,
                          int wordlen,
                          int stopbits );
```

This function is **deprecated**! Use **fossil_setbaud** instead.

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*baud_divider*

This parameter is the baud_divider computed as:

```
baud_divider = maximum_baud_rate / desired baud rate
```

where `maximum_baud_rate` is obtained through HAL **hal_get_frequencies** function.

To arrive at the closest fit value, the following round off calculation provides a good digital implementation of the above equation:

```
baud_divider = (unsigned int)
    ((((2*maximum_baud_rate) / desired_baud) + 1)
             >> 1) ;
```

This calculation first scales up the baud divider to @B1, then adds one for round off and finally scales the baud divider back to @B0 scaling with the 1 bit right shift, thereby arriving at the closest fit count for the desired baud rate.

*parity*

Enumeration type:
FOSSIL_PARITY_NO = 0
FOSSIL_PARITY_ODD = 1
FOSSIL_PARITY_EVEN = 2
FOSSIL_PARITY_MARK = 3
FOSSIL_PARITY_SPACE = 4

*wordlen*

Bits per character (7 or 8)

*stopbits*

Stop bits (1 or 2)

## Return Value

None

## Comments

If the DMA mode (send or receive mode) is enabled at **CHIP.INI**, the following port settings are not allowed:

1. 8N2 (this limitation applies only to SC1x systems)
2. 8S1
3. 8M1

## SC1x Comments

Two stop bits are only available if no parity is set.

## SC1x3/SC23 Comments

Two stop bits are only available if no parity is set or word length is equal to 7 bits.

## See Also

**fossil_setbaud** **fossil_setbaud_std**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## fossil_set_flowcontrol

Enable/disable flow control.

```
void fossil_set_flowcontrol ( int port,
                              FOS_FLOW_CTRL flowctrl );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*flowctrl*

Enumerator specifies desired type of flow control:

FOSSIL_FLOWCTRL_OFF = 0x0: No flow control
FOSSIL_FLOWCTRL_XONXOFF_SEND = 0x1:
XON/XOFF on transmit (watch for XOFF while sending)
FOSSIL_FLOWCTRL_RTSCTS = 0x2:
CTS/RTS (CTS on transmit/RTS on receive)
FOSSIL_FLOWCTRL_XONXOFF_RECV = 0x8:
XON/XOFF on receive (send XOFF when buffer near full)
FOSSIL_FLOWCTRL_XONXOFF_SEND_RECV = 0x9:
XON/XOFF used on both Tx and Rx

## Return Value

None

## Comments

Configures the flow control for a port.

Since SC12 @CHIP-RTOS 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 immediateley.  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.

## See Also

**fossil_setbaud**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_force_rts

Force RTS# to low, high or reset to normal activity

```
void fossil_force_rts ( int port, unsigned char mode );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*mode*

Mode:
RTS_LOW = 0, force RTS to low
RTS_HIGH = 1, force RTS to high
RTS_NORMAL = 2, reset to normal activity

## Return Value

None

## Comments

RTS/CTS flow control must be enabled to use this function.

## See Also

**fossil_set_flowcontrol**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_read_cts

Read level on CTS# pin

```
unsigned char fossil_read_cts ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

0, CTS# is low
1, CTS# is high

## Comments

RTS/CTS flow control must be enabled to use this function.

## See Also

**fossil_set_flowcontrol**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_user_callback

Install a Fossil User Callback Function. The User Callback function will be called when a fossil event occurs.

```
void fossil_user_callback ( int port,
                            FOSSIL_USER_CALLBACK pfFossUCB );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:

```
FOSSIL_EXT = 0  for the EXT (COM0) port (SC1x, SC1x3)
FOSSIL_COM = 1  for the COM (COM1) port (SC1x, SC1x3, SC23)
FOSSIL_SER2 =  2for the COM2 port (SC1x3, SC23)
FOSSIL_SER3 =  3for the COM3 port (SC1x3, SC23)
```

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*pfFossUCB*

Pointer to Fossil User CallBack function

## Comments

This function installs a user callback function for the specified serial port. The callback function must be very short! Long fossil callback functions could lead to character losing. To use the fossil callback functions you have to switch the corresponding serial port into the IRQ Mode (see **CHIP.INI**). In DMA Mode the callback functions do not work!

The callback function must be declared like the function below:

```
fossil_event_t far *(huge my_fossil_callback)(
    fossil_event_t far *e )
```

The callback function must be installed like below:

```
fossil_user_callback( port,
                      (FOSSIL_USER_CALLBACK)my_fossil_callback );
```

Note that only one user callback function can be installed per port. To uninstall a user callback function, simply call this function again with another function pointer or with a NULL pointer.

## See Also

**Fossil Event Data Structure**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_status_request

Get port status.

```
int fossil_status_request ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

Status code (bit field):
bit 6: FOSSIL_OUTPUT_BUFFER_EMPTY
bit 5: FOSSIL_OUTPUT_NOT_FULL
bit 4: FOSSIL_LINE_BREAK Line break detected
bit 3: FOSSIL_FRAMING_ERROR Framing error detected
bit 2: FOSSIL_PARITY_ERROR Parity error detected
bit 1: FOSSIL_OVERRUN_ERROR Set if overrun occurred on receiver.
bit 0: FOSSIL_DATA_AVAILABLE Set if data is available in receiver buffer.

The header file contains the indicated mask definitions.

## Comments

Any reported UART error flags are cleared after the read is made for this call.

For efficient receiver polling, the **fossil_getbyte** or **fossil_readblock** methods are recommended.

## See Also

**fossil_getbyte_wait** **fossil_getbyte** **fossil_readblock**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_get_driver_info

Get driver info.

```
int fossil_get_driver_info (int port,
        FossilDriverInfo_t far *drv_info);
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*drv_info*

Output parameter: Pointer to **FossilDriverInfo** data structure which will be filled by this call.

## Return Value

0x13, = sizeof(FossilDriverInfo_t)

## Comments

Get information about a serial port and driver.

## See Also

**fossil_get_bytes_in_uart**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_get_bytes_in_uart

Get number of bytes currently in UART hardware transmit buffers.

```
int fossil_get_bytes_in_uart ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

Number of transmitter bytes in the UART, pending output
-1 if UART transmitter is disabled (since SC1x version V1.20 and SC1x3 version V1.05)

## Comments

Returns the number of bytes which are currently in the UART hardware transmit buffers.

With this function you can check how many bytes are currently stored at the internal UART transmit registers. This may be necessary to know if your communication pauses because of a handshake problem. If you want to know, how many bytes are stored at the fossil driver software transmit queue, you can use **fossil_get_driver_info** function. However, fossil_get_driver_info does not check the UART transmit hardware buffers.

## See Also

**fossil_get_driver_info**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_set_rs485

Enable/Disable RS485 mode.

```
void fossil_set_rs485 ( int port, RS485_MODE mode);
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*mode*

RS 485 mode enumerator:
FOSSIL_RS485_LOWACTIVE = 0: TxEnable low active
FOSSIL_RS485_HIGHACTIVE = 1: TxEnable high active
FOSSIL_RS485_DISABLE = 2: Disable RS485 mode

## Return Value

None

## Comments

By default the RTS signals (pins) are used to enable/disable the respective transmitter. (TxEnable)

## See Also

**fossil_set_flowcontrol** **fossil_set_rs485_txenable**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_set_rs485_txenable

Select RS485 pin.

```
void fossil_set_rs485_txenable ( int port, int pin );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## SC1x Parameters

*pin*

PIO pin number, [0..13]

## SC1x3 Parameters

*pin*

PIO pin number, [0..24,26..31]
Use -1 to select the RTS pin (hardware functionality / default)

## SC23 Parameters

*pin*

PIO pin number, [0..1,9..11,13,18..23,27..28]
Use -1 to select the RTS pin (hardware functionality / default)

## Return Value

None

## Comments

By default the RTS signals (pins) are used to enable/disable the respective transmitter. (TxEnable)  This function lets you select any PIO as TxEnable (but not all make sense).  To change the default, call this function before you call the **fossil_set_rs485** function.

## See Also

**fossil_set_rs485**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_putbyte_wait

Put byte in output buffer, wait if needed.

```
int fossil_putbyte_wait ( int port, int ch );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*ch*

Byte to output

## Return Value

Status code (bit field):
bit 6: FOSSIL_OUTPUT_BUFFER_EMPTY
bit 5: FOSSIL_OUTPUT_NOT_FULL
bit 4: FOSSIL_LINE_BREAK Line break detected
bit 3: FOSSIL_FRAMING_ERROR Framing error detected
bit 2: FOSSIL_PARITY_ERROR Parity error detected
bit 1: FOSSIL_OVERRUN_ERROR Set if overrun occurred on receiver.
bit 0: FOSSIL_DATA_AVAILABLE Set if data is available in receiver buffer.

The header file contains the indicated mask definitions.

## Comments

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.)

## See Also

fossil_putbyte fossil_getbyte_wait fossil_writeblock

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## fossil_putbyte

Put byte in output buffer (without wait).

```
int fossil_putbyte ( int port, int ch );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*ch*

Byte to send

## Return Value

0 if byte was not accepted (no space in buffer)
1 if byte was placed in output buffer

## Comments

Place a byte into the transmit buffer if there is space available. Otherwise simply return with 0, without handling the transmit byte.

## See Also

**fossil_putbyte_wait** **fossil_writeblock** **fossil_getbyte**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_writeblock

Write a block of data (no wait).

```
unsigned int fossil_writeblock ( int port,
                                 const unsigned char far *buffer,
                                       unsigned int count );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:

`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*buffer*

Input parameter: Pointer to data to be transmitted

*count*

Maximum number of bytes to transfer.

## Return Value

Number of bytes transfered to transmit buffer from `buffer` .

## Comments

Write a block of data to the serial port output buffer. This function returns immediately after transfering as many bytes as **space** in transmit buffer allows.

## See Also

**fossil_readblock** **fossil_putbyte_wait** **fossil_putbyte**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_flush_output

Flush output buffer, wait until done.

```
void fossil_flush_output ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

None

## Comments

Wait for all output in the output buffer to be transmitted.

## See Also

**fossil_purge_output** **fossil_purge_input**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_purge_output

Purge output buffer and discard the data.

```
void fossil_purge_output ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

None

## Comments

Remove all data from the output buffer and discard it.

## See Also

**fossil_flush_output** **fossil_purge_input**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_send_break

Send break

```
void fossil_send_break ( int port, FOS_BREAK_SIZE break_size );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*break_size*

Enumeration type:
FOS_LONG_BREAK = 1, long break
FOS_SHORT_BREAK = 2, short break
FOS_EXTRALONG_BREAK = 3, extra long break

## Return Value

None

## Comments

A short break is a continuous Low on the TXD output for a duration of more than one frame transmission time M, where:

M = startbit + data bits (+ parity bit)+ stop bit

A long break is a continuous Low on the TXD output for a duration of more than two frame transmission times plus the transmission time for three additional bits (2M+3).

A extra long break is a continuous Low on the TXD output for a duration of more than three frame transmission times.

## See Also

**fossil_setbaud**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_enable_transmitter

Enable/Disable transmitter

```
void fossil_enable_transmitter ( int port, unsigned char mode );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*mode*

Mode: FOS_DISABLE = 0, disable transmitter, FOS_ENABLE = 1, enable transmitter

## Return Value

None

## Comments

By default the receiver is enabled.

## See Also

**fossil_enable_receiver**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_getbyte_wait

Get byte from input buffer (with wait).

```
int fossil_getbyte_wait ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

Received byte

## Comments

Reads a byte from the receiver buffer. Wait for a byte to arrive if none is available.

## See Also

**fossil_getbyte** **fossil_readblock** **fossil_putbyte_wait**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_getbyte

Get byte from input buffer (no wait).

```
int fossil_getbyte ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

-1 if no data is available in receiver buffer, else next received byte

## Comments

This function uses Fossil interrupt 0x14 Service **0x18** to read out a block of length one. This method of polling for single characters at the receiver will be considerably more efficient than the alternative method using the **fossil_status_request** and **fossil_getbyte_wait** function pair.

## See Also

**fossil_readblock** **fossil_getbyte_wait** **fossil_status_request**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_peek_input

Peek if next byte is available.

```
int fossil_peek_input ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

-1 if no character available,
Else next character in receiver buffer.

## Comments

Returns the next byte available in the receive buffer, without removing it from the buffer.

## See Also

**fossil_getbyte_wait** **fossil_getbyte** **fossil_status_request**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_readblock

Read block of data (no wait).

```
unsigned int fossil_readblock ( int port,
                                unsigned char far *buffer,
                                unsigned int count );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:

`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*buffer*

Output parameter: Pointer to buffer to put received bytes into.

*count*

Size of `buffer` , maximum number of bytes to transfer.

## Return Value

Number of bytes transfered

## Comments

Read up to a specified number of bytes from a serial port input **queue** and return immediately.

## See Also

**fossil_writeblock** **fossil_getbyte_wait** **fossil_getbyte**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_purge_input

Purge input buffer and discard the data.

```
void fossil_purge_input ( int port );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

## Return Value

None

## Comments

Remove all data from the receive buffer and discard the data.

## See Also

**fossil_flush_output** **fossil_purge_output**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_enable_receiver

Enable/Disable receiver

```
void fossil_enable_receiver ( int port, unsigned char mode );
```

## Parameters

*port*

Port number:
This can be either a numeric value or one of the following constants:
`FOSSIL_EXT = 0` for the EXT (COM0) port (SC1x, SC1x3)
`FOSSIL_COM = 1` for the COM (COM1) port (SC1x, SC1x3, SC23)
`FOSSIL_SER2 = 2`for the COM2 port (SC1x3, SC23)
`FOSSIL_SER3 = 3`for the COM3 port (SC1x3, SC23)

Besides the numbers for the IPC@CHIP®'s internal ports, a number of an external serial port can be passed if such a port is installed (see **fossil_register_external_port**).

*mode*

Mode: FOS_DISABLE = 0, disable receiver, FOS_ENABLE = 1, enable receiver

## Return Value

None

## Comments

This function is useful when using the SM35 as RS485 adapter. Use this function to disable the receiver before transmitting data. Wait until all data is send and reenable the receiver again. This way you don't receive your own transmitted data.

By default the receiver is enabled.

## See Also

**fossil_enable_transmitter**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## fossil_register_external_port

This function registers an external port, i.e. a serial port which is handled by an external driver program. The driver program must provide several callback functions, which are called when another program accesses the assigned port number.
This mechanism can be used to integrate additional real serial ports (additional UART hardware) or virtual serial ports, which are for example forwarded via USB, into the RTOS.

```
short fossil_register_external_port ( short port,
                                      FossilExternalPortCallbacks
                                        *callbacks );
```

## Parameters

*port*

Designated port number, -1 if an arbitrary port number is to be assigned

*callbacks*

Pointer to structure holding pointers to the several callback functions. For details see **FossilExternalPortCallbacks**.

## Return Value

On success, port number assigned to the external port driver
-1, if the designated port number is not free or belongs to one the the RTOS' standard serial ports
-2, if the maximum number of external port drivers has already been registered.
-3, if either the argument `callbacks` or `callbacks->unsupported_interrupt` is NULL

## Example

The following example shows how to register an external port driver. It is assumed that the callback functions are named `myFossil_*`.

```
int myFossil_init(int port)
{
  ... // Implementation
}

... // Other callback functions

unsigned int myUnsupported_interrupt(unsigned ax, unsigned bx, unsigned cx,
                                     unsigned dx, unsigned es, unsigned di)
{
```

```
  ... // Implementation
}

void main(void)
{
  FossilExternalPortCallbacks callbacks;
  short portNumber;
  ...

  callbacks.init = myFossil_init;
  ... // Other callback functions
  callbacks.set_rs485 = NULL;  // Unimplemented function
  callbacks.unsupported_interrupt = myUnsupported_interrupt;

  portNumber = fossil_register_external_port(-1, &callbacks);
  if(portNumber < 0)
    ...  // Error handling

  ... // Program runs

  portNumber = fossil_unregister_external_port(portNumber);
  if(portNumber < 0)
    ...  // Error handling
}
```

## See Also

**fossil_unregister_external_port**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.
Please do not use the software interrupt version of this function directly, since parts of the functionality are implemented within the C-library.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.20 | V1.20 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.13 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## fossil_unregister_external_port

This function unregisters an external port.

```
short fossil_unregister_external_port ( short port );
```

## Parameters

*port*

Number of the external port

## Return Value

0 on success
-1, if no external port has been registered under the given number

## See Also

fossil_register_external_port

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.
Please do not use the software interrupt version of this function directly, since parts of the functionality are implemented within the C-library.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.20 | V1.20 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.13 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS C Library - Helper Functions

---

## helper_strout

String output function

```
void helper_strout ( const char far *str );
```

## Parameters

*str*

Zero terminated output string to print over STDOUT

## Return Value

-- none --

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Helper Functions

## helper_chrout

Character output function

```
int helper_chrout ( char ch, int zero );
```

## Parameters

*ch*

Character to print over STDOUT

*zero*

Zero-Mode
if ch>=10 zero is ignored.
if ch<10 zero is ignored but the character printed out is ch + 48.
if ch==0 and zero==0 character print is ch + 48.
if ch==0 and zero < 0 no character is printed
if ch==0 and zero > 0 a space character is printed.

## Return Value

Value in zero is returned

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## helper_dezout

Character output function

```
void helper_dezout ( unsigned int x, int zero );
```

## Parameters

*x*

Decimal value to print over STDOUT

*zero*

Zero-Mode
if zero==0, leading zeros are printed as zeros.
if zero<0, leading zeros are not printed
if zero>0, leading zeros are printed as spaces.

## Return Value

-- none --

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS C Library - Helper Functions

---

## helper_hexdig

Hexadecimal digit output function

```
void helper_hexdig ( unsigned char dig );
```

## Parameters

*dig*

Hexadecimal digit (nibble) to print over STDOUT

## Return Value

-- none --

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS C Library - Helper Functions

---

## helper_hexout

Hexadecimal integer output function

```
void helper_hexout ( unsigned int hex );
```

## Parameters

*hex*

Hexadecimal integer to print over STDOUT

## Return Value

-- none --

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Helper Functions

---

## helper_printf

Formatted console output function like the standard library `printf` function.

```
int helper_printf ( const char *fmt, ... );
```

## Parameters

*`fmt`*

Format specifier string. Note: Floating point types are not suppported by this implementation.

## Return Value

Number of characters output to console. (If @Chip-RTOS version does not support this function, then this return value will always be zero.)

## Comments

Using this function instead of the standard library `printf` will reduce a program's memory requirements.

**WARNING for small and medium memory models:**

Pointers which are passed to this function after the first declared argument must be explicitly cast as type "far". (Compiler has no clue about these additional arguments so your help is needed.) For example:

```
helper_printf(format, (char far *)string);
```

The exception to this rule are pointers for display with the %p format. These are taken as long by default (regardless of your program's memory model) and can be specified explicitly as near with the N input format modifier, in which case you should only pass a 16 bit pointer. See your compiler's `printf` documentation for more details.

**CAUTION:**

When this function is used with SC1x @Chip-RTOS versions prior to version 1.20 a stack buffer will overflow if the total number of characters headed to the console exceeds 179 characters. For subsequent @Chip-RTOS versions there is no such limitation.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## helper_sprintf

String formatting like the standard library `sprintf` function.

```
int helper_sprintf (char *buffer,
                    const char *fmt, ... );
```

## Parameters

*buffer*

Output parameter: Formatted text will be written into this buffer.

*fmt*

Format specifier string. Note: Floating point types are not suppported by this implementation.

## Return Value

Number of characters output to `buffer` not counting the final NIL string terminating character. (If @Chip-RTOS version does not support this function, then this return value will always be zero.)

## Comments

Using this function instead of the standard library `sprintf` will reduce a program's memory requirements.

**WARNING for small and medium memory models:**

Pointers which are passed to this function after the first two declared arguments must be explicitly cast as type "far". (Compiler has no clue about these additional arguments so your help is needed.) For example:

```
helper_sprintf(buffer, format, (char far *)string);
```

The exception to this rule are pointers for display with the %p format. These are taken as long by default (regardless of your program's memory model) and can be specified explicitly as near with the N input format modifier, in which case you should only pass a 16 bit pointer. See your compiler's `sprintf` documentation for more details.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.03 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## helper_release_rtos_mem

Release allocated RTOS memory (Int21h function 49h)

```
int helper_release_rtos_mem(unsigned int seg);
```

## Parameters

*seg*

Segment of the memory area to be released

## Return Value

0 success
-1 failed

## See Also

helper_alloc_rtos_mem  helper_alloc_rtos_fast_mem

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## helper_assign_mem

Assign memory block ownership to a specific task.

```
int helper_assign_mem(void far *mem_block, unsigned int taskid);
```

## Parameters

*mem_block*

Pointer to RTOS system memory block.

*taskid*

Task ID of the **task** to assign the ownership of this memory block. This parameter may be set to zero to assign ownership to the current task which is making this call.

## Return Value

error code
0: Success
-1: Invalid task ID parameter, `taskid`
-38: Invalid RTOS memory pointer, `mem_block`
.

## Comments

The referenced memory block must be one that originated from one of the RTOS **system memory** allocators. This includes those blocks allocated by either **helper_alloc_rtos_mem** or **helper_alloc_rtos_fast_mem** API.

This function can be useful to control the lifespan of a given block of memory. After the specified task has been assigned ownership of this memory block, this memory block will automatically be released when that task is **deleted** from the system if the block had not already been released.

A task's stack space allocated from RTOS memory can safely be released in this manner.

## See Also

**helper_alloc_rtos_mem** **helper_alloc_rtos_fast_mem** **helper_release_rtos_mem**
**RTX_Delete_Task**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.23 | V1.23 | V1.23 | V1.12 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.09 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## helper_check_license

Check for installed license.

```
int helper_check_license(unsigned char id,
                         unsigned char far *secret,
                         unsigned short length);
```

## Parameters

*id*

ID of license key

*secret*

Pointer to secret key

*length*

Length of secret key in bytes

## Return Value

error code
- 0: Success, found a valid license
- -1: Error, no valid license for specified ID found
- -2: Error, reading product data failed
- -3: Error, MD5 functions failed

## Comments

The function validates that a license for the specified ID is installed correctly. Computers without a valid license can be rejected by calling this function.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.10 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.10 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## helper_load_library

Load and link a dynamic link library.

```
int helper_load_library(char far * libName,
                        void far * pFpl,
                        int * pFplSize,
                        int maxFplSize);
```

## Parameters

*libName*

filename of the DLL (with extention!)

*pFpl*

Pointer to the function pointer list where the function will store the pointers to the function which will be exported by the DLL.

*pFplSize*

Size of the result fpl (from the DLL) in bytes (Output parameter).

*maxFplSize*

Maximal size of function pointer list in bytes.

## Return Value

error code
CHIPDLL_ERR_OK: Success, DLL was loaded and functions were exported
CHIPDLL_ERR_COULDNTLD: DLL could not be loaded
CHIPDLL_ERR_NOTFOUND: The dll was not found
DLL specifix error code, defined by the DLL developer.

## Comments

If the DLL is not load yet the function loads and starts the DLL.

The order of the exported functions in the function pointer list is equal to the order in the DLLs project export function declaration.

Example:

```
struct MYDLL
{
  int  (*myFirstDllFunction)(int k);
  long (*mySecondDllFunction)(char *s);
};

int result;
int fplSize;
struct MYDLL myDll;

// Sign in to the DLL. When the DLL is not loaded yet it will be loaded automatically.
result = helper_load_library( "myLibrary.dll", &myDll, &fplSize, sizeof(myDll) );
if (result != CHIPDLL_ERR_OK)
{
  // Error while load DLL!
  return -1;
}

if ( fplSize < sizeof(myDll) )
{
  // Not all function linked! Maybe it is an older
  // version of the DLL ?
  helper_unload_library( "myLibrary.dll" );
  return -2;
}

// Now the DLLs function could be called
myDll.myFirstDllFunction(5);
myDll.mySecondDllFunction("hello");

// When the DLL is no more required, the application should sign off
// When no more application is signed in, it will be removed from memory.
helper_unload_library( "myLibrary.dll" );
```

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.10 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.13 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Helper Functions

## helper_unload_library

Unload a dynamic link library.

```
int helper_unload_library(char far * libName);
```

## Parameters

*libName*

filename of the DLL (with extention!)

## Return Value

error code
CHIPDLL_ERR_OK: Success, DLL was unloaded successfully

## Comments

If there is no more application which signed on to the DLL it will be removed from memory within this call.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.10 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.13 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS C Library - Hardware API

---

Header File: `#include <HWAPI.H>`

**PFE**

**HAL**

## PFE - Pin Function Enabler Functions

| | |
|---|---|
| pfe_enable_bus | Enable data bus |
| pfe_enable_adr | Enable non-multiplexed address bus |
| pfe_enable_pio | Enable programmable I/O pins |
| pfe_get_pio_state | Get programmable PIO pins state |
| pfe_get_pio_state2 | Get programmable PIO pins state (PIO16-31) |
| pfe_enable_pcs | Enable programmable chip selects |
| pfe_enable_int | Enable external interrupt requests |
| pfe_enable_timer | Enable external timer inputs/outputs |
| pfe_set_edge_level_intr_mode | Set edge/level interrupt mode |
| pfe_enable_pwd | Enable PWD mode |
| pfe_enable_external_dma | Enable external DMA |
| pfe_enable_int_cascade_mode | Enable INT0/INTA cascade mode |
| pfe_set_wait_states | Set wait states for PCS0-3 |
| pfe_set_wait_states2 | Set wait states for PCS5-6, respectively PCS4-7 |
| pfe_set_wait_states_ucs | Set wait states for UCS |
| pfe_enable_bus_signals | Enable bus signals (ARDY, HOLD, WRL/WRH, A0/BHE) |
| pfe_get_vectors | Get Hardware API function pointers |

## HAL - Hardware Abstraction Layer Functions

| | |
|---|---|
| -- Data Bus I/O -- | |
| hal_read_bus | **Deprecated!** Read data bus |
| hal_write_bus | **Deprecated!** Write data bus |
| hal_readblock_bus | **Deprecated!** Block read data bus |

| | |
|---|---|
| **hal_writeblock_bus** | **Deprecated!** Block write data bus |
| -- PIO -- | |
| **hal_read_pios** | Read programmable I/O pins |
| **hal_read_pios2** | Read programmable I/O pins |
| **hal_write_pios** | Write programmable I/O pins |
| **hal_write_pios2** | Write programmable I/O pins |
| **hal_read_pio** | Read specific I/O pin |
| **hal_write_pio** | Write to specific I/O pin |
| -- Interrupts -- | |
| **hal_install_isr** | Install Interrupt Service Routine |
| **hal_install_rtx_isr** | Install RTX Interrupt Service Routine |
| **hal_give_eoi** | Issue EOI to interrupt controller |
| **hal_mask_int** | Mask/Unmask Interrupts |
| -- Timer Operation -- | |
| **hal_init_timer** | Initialize timer settings |
| **hal_init_timer_ext** | Initialize timer settings Ext |
| **hal_start_timer** | Start timer |
| **hal_stop_timer** | Stop timer |
| **hal_read_timer_count** | Read timer count |
| **hal_write_timer_count** | Write timer count |
| **hal_set_duty_cycle_waveform** | Set timer duty cycle waveform |
| -- Watchdog -- | |
| **hal_set_watchdog_mode** | Get/Set watchdog mode |
| **hal_refresh_watchdog** | Refresh watchdog |
| -- DMA Control -- | |
| **hal_start_dma** | Start DMA mode |
| **hal_stop_dma** | Stop DMA transfer |
| **hal_get_dma_info** | Get DMA info |
| -- Non-Volatile Data -- | |
| **hal_init_non_volatile_data** | Initialize/Restore non-volatile data |
| **hal_save_non_volatile_data** | Save non-volatile data |
| **hal_get_reboot_reason** | Get reboot reason |
| **hal_install_pfaildone_callback** | Install PowerfailDone callback |
| -- other functions -- | |
| **hal_get_frequencies** | Get frequencies |

List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## pfe_enable_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 mask bit for bi-directional data bus lines (read/write) should be set to '1'.

```
void pfe_enable_bus ( unsigned short mask, unsigned char ale );
```

## SC1x Parameters

*mask*

Data bus 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)

*ale*

0: Disable ALE, 1: Enable ALE

## SC1x3 Parameters

*mask*

Data bus 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 15 = 0: Data bus bit 15 is input, 1: is output

*ale*

Don't care

## Return Value

-- none --.

## SC1x Comments

Pins used:
ALE, AD[0..7], RD#, WR#
Excluded pins:

if ALE is used, then PCS0# is not available.

## SC1x3 Comments

Pins used:
A[0..22], D[0..15], RD#, WR#

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_enable_adr

The IPC@CHIP® has non-multiplexed address bit outputs, e.g. A0 on SC1x or A23 on SC1x3. The enabling of one of these pins is done here.

```
void pfe_enable_adr ( unsigned char adr );
```

## SC1x Parameters

*adr*

0: Enables A0
1: Enables A1
2: Enables A2

## SC1x3 Parameters

*adr*

23: Enables A23

## Return Value

-- none --.

## SC1x 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.

## SC1x3 Comments

Used pins:
A23
Excluded pins:
If A23 is enabled then PIO26 is not available

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_enable_pio

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 **pfe_enable_pcs** for PIO2).

```
void pfe_enable_pio ( unsigned short pio, unsigned char mode );
```

## SC1x Parameters

*pio*
PIO pin number, [0..13]

*mode*
Mode
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

## SC1x3 Parameters

*pio*
PIO pin number, [0..24, 26..31]

*mode*
Mode
1,2 = Input with pullup
4 = Output init value = High
5 = Output init value = Low

## SC23 Parameters

*pio*
PIO pin number, [0..2, 9..13, 18..23, 27..28, 31]

*mode*
Mode
1,2,3 = Input with pullup/down (PIO1/PIO2=pulldown, all other=pullup)
4 = Output init value = High
5 = Output init value = Low

## Return Value

-- none --.

## Comments

Used pins:
PIO[0..x]
Excluded pins:
All other functionality on the selected PIO pin.

## See Also

hal_read_pio hal_write_pio I2C_init
hal_read_pios hal_write_pios

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## pfe_get_pio_state

Get programmable PIO pins state.

```
void pfe_get_pio_state ( unsigned int far * wPio,
                         unsigned int far * wInp,
                         unsigned int far * wOut );
```

## SC1x Parameters

*wPio*

Output parameter:
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)

*wInp*

Output parameter:
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)

*wOut*

Output parameter:
Bit 0 = 1: PIO0 is output
Bit 1 = 1: PIO1 is output
:
Bit 13 = 1: PIO13 is output
Bit 14..15 not used (for future extensions)

## SC1x3 Parameters

*wPio*

Output parameter:
Bit 0 = 1: PIO0 is PIO
Bit 1 = 1: PIO1 is PIO
:
:
Bit 15 = 1: PIO15 is PIO

*wInp*

Output parameter:
Bit 0 = 1: PIO0 is input
Bit 1 = 1: PIO1 is input
:
Bit 15 = 1: PIO15 is input

*wOut*

Output parameter:
Bit 0 = 1: PIO0 is output
Bit 1 = 1: PIO1 is output
:
Bit 15 = 1: PIO15 is output

## SC23 Parameters

*wPio*

Output parameter:
Bit 0 = 1: PIO0 is PIO
Bit 1 = 1: PIO1 is PIO
Bit 2 = 1: PIO2 is PIO
Bit 9 = 1: PIO9 is PIO
Bit 10 = 1: PIO10 is PIO
Bit 11 = 1: PIO11 is PIO
Bit 12 = 1: PIO12 is PIO
Bit 13 = 1: PIO13 is PIO

*wInp*

Output parameter:
Bit 0 = 1: PIO0 is input
Bit 1 = 1: PIO1 is input
Bit 2 = 1: PIO2 is input
Bit 9 = 1: PIO9 is input
Bit 10 = 1: PIO10 is input
Bit 11 = 1: PIO11 is input
Bit 12 = 1: PIO12 is input
Bit 13 = 1: PIO13 is input

*wOut*

Output parameter:
Bit 0 = 1: PIO0 is output
Bit 1 = 1: PIO1 is output
Bit 2 = 1: PIO2 is output
Bit 9 = 1: PIO9 is output
Bit 10 = 1: PIO10 is output
Bit 11 = 1: PIO11 is output
Bit 12 = 1: PIO12 is output
Bit 13 = 1: PIO13 is output

## Return Value

-- none --.

## Comments

The value output to `wPio` has bits set to one to indicate PIO is configured as an I/O pin.

The value output to `wInp` has bits set to one to indicate PIO which are configured as inputs.

The value output to `wOut` has bits set to one to indicate PIO which are configured as outputs.

## See Also

hal_read_pio hal_write_pio
hal_read_pio hal_write_pio pfe_get_pio_state2
hal_read_pios hal_write_pios

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_get_pio_state2

Get programmable PIO pins state (PIOs 16-31).

```
void pfe_get_pio_state2 ( unsigned int far * wPio,
                          unsigned int far * wInp,
                          unsigned int far * wOut );
```

### SC1x3 Parameters

*wPio*

Output parameter:

Bit 0 = 1: PIO16 is PIO
Bit 1 = 1: PIO17 is PIO
:
Bit 15 = 1: PIO31 is PIO

*wInp*

Output parameter:

Bit 0 = 1: PIO16 is input
Bit 1 = 1: PIO17 is input
:
Bit 15 = 1: PIO31 is input

*wOut*

Output parameter:

Bit 0 = 1: PIO16 is output
Bit 1 = 1: PIO17 is output
:
Bit 15 = 1: PIO31 is output

### SC23 Parameters

*wPio*

Output parameter:

Bit 2 = 1: PIO18 is PIO
Bit 3 = 1: PIO19 is PIO
Bit 4 = 1: PIO20 is PIO
Bit 5 = 1: PIO21 is PIO
Bit 6 = 1: PIO22 is PIO
Bit 7 = 1: PIO23 is PIO
Bit 11 = 1: PIO27 is PIO
Bit 12 = 1: PIO28 is PIO
Bit 15 = 1: PIO31 is PIO

*wInp*

Output parameter:

Bit 2 = 1: PIO18 is input
Bit 3 = 1: PIO19 is input
Bit 4 = 1: PIO20 is input
Bit 5 = 1: PIO21 is input
Bit 6 = 1: PIO22 is input
Bit 7 = 1: PIO23 is input
Bit 11 = 1: PIO27 is input
Bit 12 = 1: PIO28 is input
Bit 15 = 1: PIO31 is input

*wOut*

Output parameter:

Bit 2 = 1: PIO18 is output
Bit 3 = 1: PIO19 is output
Bit 4 = 1: PIO20 is output
Bit 5 = 1: PIO21 is output
Bit 6 = 1: PIO22 is output
Bit 7 = 1: PIO23 is output
Bit 11 = 1: PIO27 is output
Bit 12 = 1: PIO28 is output
Bit 15 = 1: PIO31 is output

## Return Value

-- none --.

## Comments

The value output to `wPio` has bits set to one to indicate PIO is configured as an I/O pin.

The value output to `wInp` has bits set to one to indicate PIO which are configured as inputs.

The value output to `wOut` has bits set to one to indicate PIO which are configured as outputs.

## See Also

**hal_read_pio** **hal_write_pio** **pfe_get_pio_state**
**hal_read_pios** **hal_write_pios**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## pfe_enable_pcs

Enable chip selects PCSx#

```
void pfe_enable_pcs ( unsigned char pcs );
```

## SC1x Parameters

*pcs*

Chip select number:

0: Enable PCS0#, active when I/O address between 000h..0FFh
1: Enable PCS1#, active when I/O address between 100h..1FFh
2: Enable PCS2#, active when I/O address between 200h..2FFh
3: Enable PCS3#, active when I/O address between 300h..3FFh
4: N/A
5: Enable PCS5#, active when I/O address between 500h..5FFh
6: Enable PCS6#, active when I/O address between 600h..6FFh

## SC1x3 Parameters

*pcs*

Chip select number:

0: don't care (PCS0# is no PIO), PCS0# is active when I/O address between 000h..1FFh
1: don't care (PCS1# is no PIO), PCS1# is active when I/O address between 200h..3FFh
2: don't care (PCS2# is no PIO), PCS2# is active when I/O address between 400h..5FFh
3: don't care (PCS3# is no PIO), PCS3# is active when I/O address between 600h..7FFh
4: Enable PCS4#, active when I/O address between 800h..9FFh
5: Enable PCS5#, active when I/O address between A00h..BFFh
6: Enable PCS6#, active when I/O address between C00h..DFFh
7: Enable PCS7#, active when I/O address between E00h..FFFh

## Return Value

-- none --.

## SC1x 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

## SC1x3 Comments

Used pins:
PCS[0..7]#

Excluded pins:
If PCS4#: PIO4
If PCS5#: PIO3
If PCS6#: PIO2
If PCS7#: PIO5

## See Also

**pfe_set_wait_states**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_enable_int

Enable external interrupt requests INTx

```
void pfe_enable_int ( unsigned char irq );
```

## SC1x Parameters

*irq*

External Interrupt number:

0: Enable INT0
1: N/A
2: Enable INT2
3: Enable INT3
4: Enable INT4
5: Enable INT5
6: Enable INT6

## SC1x3/SC23 Parameters

*irq*

External Interrupt number:

0: N/A
1: Enable INT1
2: N/A
3: Enable INT3
4: N/A
5: Enable INT5
6-14: N/A
15: Enable PFI

## Return Value

-- none --.

## SC1x 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 I2C-Bus pins, SPI
if INT6: PIO0, DRQ1, default I2C-Bus pins, SPI

## SC1x3 Comments

Used pins:
INT1, INT3, INT5, PFI

Excluded pins:
if INT5: PIO30
if PFI: PIO9

## SC23 Comments

Used pins:
INT1, INT3, INT5, PFI

Excluded pins:
if INT5: PIO0 / TMRIN1
if INT3: PIO1 / TMROUT1
if INT1: PIO2
if PFI: PIO9

## See Also

**pfe_set_edge_level_intr_mode** **hal_install_isr** **hal_install_rtx_isr**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_enable_timer

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.

```
void pfe_enable_timer ( unsigned short mode );
```

## Parameters

*mode*

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]

## SC1x Comments

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

## SC1x3 Comments

Excluded pins:
if TMRIN0: PIO11
if TMRIN1: PIO0
if TMROUT0: PIO10
if TMROUT1: PIO1

## SC23 Comments

Excluded pins:
if TMRIN0: PIO11
if TMRIN1: PIO0 / INT5
if TMROUT0: PIO10
if TMROUT1: PIO1 / INT3

## See Also

**hal_init_timer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_set_edge_level_intr_mode

Set edge/level interrupt mode for INTx.

```
void pfe_set_edge_level_intr_mode ( unsigned char mode,
                                    unsigned short mask );
```

## Parameters

*mode*

1 = level-sensitive interrupt
0 = edge-triggered interrupt

## SC1x Parameters

*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

## SC1x3/SC23 Parameters

*mask*

Bits set to designate interrupts affected:
- Bit 0 = don't care
- Bit 1 = INT1
- Bit 2 = don't care
- Bit 3 = INT3
- Bit 4 = don't care
- Bit 5 = INT5#
- Bit 6..15 = don't care

## Return Value

-- none --.

## SC1x 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.

Level-sensitive mode for INT5 / INT6 is not supported. The INT5 / INT6 interrupts operate only in edge-triggered mode.

In level-sensitive mode all interrupt lines are active high.
In edge-triggered mode all interrupt lines act on the low-to-high transition.

## SC1x3/SC23 Comments

Default for INT3 interrupt is level-triggered mode. Default for INT1 and INT5# interrupts is edge-triggered mode. In each case (edge or level) the interrupt pins must remain high until they are acknowledged.

Edge-sensitive mode for INT3 is not allowed if the USB controller is used, because the INT3 interrupt is shared with the USB interrupt.

In level-sensitive mode all interrupt lines except INT5# are active high. INT5# is active low.
In edge-triggered mode all interrupt lines except INT5# act on the low-to-high transition. INT5# acts on the high-to-low transition.

## See Also

**pfe_enable_int**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_enable_pwd

Enable Pulse Width Demodulation (PWD)

```
void pfe_enable_pwd ( void );
```

## Parameters

-- none --

## 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

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_enable_external_dma

Enables DRQ pin to start DMA transfer.

```
int pfe_enable_external_dma ( unsigned char channel );
```

## SC1x Parameters

*channel*
DRQ channel:
0 = DRQ0
1 = DRQ1

## SC1x3 Parameters

*channel*
DRQ channel:
0 = DRQ0
1 = DRQ1
2 = DRQ2
3 = DRQ3

## Return Value

0: no error
-1: invalid DRQ channel
-2: DMA channel is used for serial interface

## Comments

For using external DMA you have to disable the serial DMA mode. Do this with a **CHIP.INI entry**. COM uses DRQ1, EXT uses DRQ0.

## SC1x Comments

Used pins:
DRQ[0..1]

Excluded pins:
If DRQ0: PIO1, INT5, default I²C-Bus pins, SPI
If DRQ1: PIO0, INT6, default I²C-Bus pins, SPI

## SC1x3 Comments

Used pins:
DRQ[0..3]

Excluded pins:
If DRQ0: PIO12
If DRQ1: PIO29
If DRQ2: PIO7
If DRQ3: PIO8

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_enable_int_cascade_mode

Enable INT0 / INTA cascade mode

```
void pfe_enable_int_cascade_mode ( void );
```

## Parameters

-- none --

## Return Value

-- none --.

## Comments

To install a service interrupt routine in cascade mode, the **hal_install_isr** function 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 with **hal_give_eoi** (INT0).

Used pins:
INT0, INTA#

Excluded pins:
PIO13, TMROUT0, INT0 in normal mode, PIO6, PCS2#, PWD, SPI, hw flow control serial port 1

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | n/a | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## pfe_set_wait_states

Set wait states for programmable chip selects PCS0#-PCS3#

```
void pfe_set_wait_states ( PFE_WAIT_STATES wait_states );
```

## SC11/SC13 Parameters

*wait_states*
Enumerator specifies desired number of wait states:

```
typedef enum {
      PFE_0_WAIT = 0x0,
      PFE_1_WAIT = 0x1,
      PFE_2_WAIT = 0x2,
      PFE_3_WAIT = 0x3,
      PFE_5_WAIT = 0x8,
      PFE_7_WAIT = 0x9,
      PFE_9_WAIT = 0xA,
      PFE_15_WAIT = 0xB
      PFE_24_WAIT = 0x10,
      PFE_41_WAIT = 0x11,
      PFE_74_WAIT = 0x12,
      PFE_123_WAIT = 0x13,
      PFE_157_WAIT = 0x18,
      PFE_175_WAIT = 0x19,
      PFE_201_WAIT = 0x1A,
      PFE_255_WAIT = 0x1B
} PFE_WAIT_STATES ;
```

## SC12/SC1x3 Parameters

*wait_states*
Enumerator specifies desired number of wait states:

```
typedef enum {
      PFE_0_WAIT = 0x0,
      PFE_1_WAIT = 0x1,
      PFE_2_WAIT = 0x2,
      PFE_3_WAIT = 0x3,
      PFE_5_WAIT = 0x8,
      PFE_7_WAIT = 0x9,
      PFE_9_WAIT = 0xA,
      PFE_15_WAIT = 0xB
} PFE_WAIT_STATES ;
```

## Return Value

-- none --.

## Comments

Set wait states for programmable chip selects PCS0#-PCS3#.

One wait state is defined as "1/clock rate".
*Example:* The SC12 runs at 20 MHz. So one wait state has a duration of 50 ns on SC12.

## SC12/SC1x3 Comments

Default for SC12/SC123/SC143 PCS0#-PCS3# is 15 wait states.
Only 0-15 waitstates can be inserted.

## SC11/SC13 Comments

Default for SC11/SC13 PCS0#-PCS3# is 41 wait states.

## See Also

**pfe_enable_pcs** **pfe_set_wait_states2**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_set_wait_states2

Set wait states for programmable chip selects PCS5#-PCS6#, respectively PCS4#-7#

```
SC11/SC13:void pfe_set_wait_states2 ( PFE_WAIT_STATES wait_states );
SC1x3:    void pfe_set_wait_states2 ( PFE_WAIT_STATES2 wait_states );
```

## SC1x3 Parameters

*wait_states*

Enumerator specifies desired number of wait states:

```
typedef enum {
      PFE_0_WAIT2 = 0x00,
      PFE_1_WAIT2 = 0x01,
      PFE_2_WAIT2 = 0x02,
      PFE_3_WAIT2 = 0x03,
      PFE_4_WAIT2 = 0x08,
      PFE_5_WAIT2 = 0x09,
      PFE_6_WAIT2 = 0x0A,
      PFE_7_WAIT2 = 0x0B,
      PFE_8_WAIT2 = 0x10,
      PFE_9_WAIT2 = 0x11,
      PFE_10_WAIT2= 0x12,
      PFE_11_WAIT2= 0x13,
      PFE_12_WAIT2= 0x18,
      PFE_13_WAIT2= 0x19,
      PFE_14_WAIT2= 0x1A,
      PFE_15_WAIT2= 0x1B,
} PFE_WAIT_STATES2 ;
```

## SC11/SC13 Parameters

*wait_states*

Enumerator specifies desired number of wait states:

```
typedef enum {
      PFE_0_WAIT = 0x00,
      PFE_1_WAIT = 0x01,
      PFE_2_WAIT = 0x02,
      PFE_3_WAIT = 0x03,
      PFE_5_WAIT = 0x08,
      PFE_7_WAIT = 0x09,
      PFE_9_WAIT = 0x0A,
      PFE_15_WAIT = 0x0B,
      PFE_24_WAIT = 0x10,
```

```
        PFE_41_WAIT = 0x11,
        PFE_74_WAIT = 0x12,
        PFE_123_WAIT = 0x13,
        PFE_157_WAIT = 0x18,
        PFE_175_WAIT = 0x19,
        PFE_201_WAIT = 0x1A,
        PFE_255_WAIT = 0x1B
} PFE_WAIT_STATES ;
```

## Return Value

-- none --.

## SC11/SC13 Comments

Set wait states for programmable chip selects PCS5#-PCS6#.
Default for SC11/SC13 PCS5#-PCS6# is 15 wait states.

One wait state is defined as "1/clock rate".
*Example:* The SC13 runs at 40 MHz. So one wait state has a duration of 25 ns on SC13.

## SC1x3 Comments

Set wait states for programmable chip selects PCS4#-PCS7#.
Default for SC123/SC143 PCS4#-PCS7# is 15 wait states.

## See Also

**pfe_enable_pcs** **pfe_set_wait_states**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_set_wait_states_ucs

Set wait states for external memory chip select UCS#

```
void pfe_set_wait_states_ucs ( PFE_WAIT_STATES2 wait_states );
```

## Parameters

*wait_states*

Enumerator specifies desired number of wait states:

```
typedef enum {
      PFE_0_WAIT2 = 0x00,
      PFE_1_WAIT2 = 0x01,
      PFE_2_WAIT2 = 0x02,
      PFE_3_WAIT2 = 0x03,
      PFE_4_WAIT2 = 0x08,
      PFE_5_WAIT2 = 0x09,
      PFE_6_WAIT2 = 0x0A,
      PFE_7_WAIT2 = 0x0B,
      PFE_8_WAIT2 = 0x10,
      PFE_9_WAIT2 = 0x11,
      PFE_10_WAIT2= 0x12,
      PFE_11_WAIT2= 0x13,
      PFE_12_WAIT2= 0x18,
      PFE_13_WAIT2= 0x19,
      PFE_14_WAIT2= 0x1A,
      PFE_15_WAIT2= 0x1B,
} PFE_WAIT_STATES2 ;
```

## Return Value

-- none --.

## Comments

Set wait states for external memory chip select UCS#.
Default for SC123/SC143 UCS# is 9 wait states.

One wait state is defined as "1/clock rate".
*Example:* The SC1x3 runs at 96 MHz. So one wait state has a duration of 10,4166 ns.

## See Also

**pfe_enable_pcs**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_enable_bus_signals

Activate the bus signals ARDY, HOLD, WRL/WRH and A0/BHE

```
void pfe_enable_bus_signals ( PFE_BUS_SIGNALS signals );
```

## Parameters

*signals*

Signal mask:

```
typedef enum {
      PFE_SIGNAL_ARDY = 0x01,
      PFE_SIGNAL_HOLD = 0x02,
      PFE_SIGNAL_WRL_WRH = 0x04,
      PFE_SIGNAL_A0_BHE = 0x08
      } PFE_BUS_SIGNALS ;
```

## Return Value

-- none --.

## Comments

Used pins:
ARDY, HOLD

Excluded pins:
If ARDY: PIO6
If HOLD: PIO17
If WRL/WRH: A0/BHE
If A0/BHE: WRL/WRH

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | n/a |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.06 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## pfe_get_vectors

Get the Function Pointers of Hardware API functions. So the functions can be called direct (without software interrupt). This is much faster.

```
void pfe_get_vectors ( HWAPI_VECTORS far *vectors );
```

## Parameters

*vectors*

Enumerator specifies desired number of wait states:

```
struct HWAPI_VECTORS_
{
      int size;
      READ_PIOS readPios;
      WRITE_PIOS writePios;
}
HWAPI_VECTORS ;
```

where

```
typedef void (far *WRITE_PIOS) (unsigned short value) ;
typedef unsigned short (far *READ_PIOS) (void) ;
```

## Return Value

-- none --.

## Comments

The `size` element of the `HWAPI_VECTORS` structure is set to 2 within this library API interface function. The two output vectors will be placed in the `writePios` and `readPios` elements.

The prototypes for the two indirectly accessed API functions are

```
void writePios (unsigned short value) ;
unsigned short readPios (void) ;
```

where input parameter `value` and return value from `readPios` both have format:
B0..B13: PIO value
B14...B15: -- not used --

## See Also

hal_read_pio hal_write_pio
hal_read_pios hal_write_pios

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | V1.00 | n/a | n/a |

This API List
List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---

## hal_read_bus

Read single byte from data bus. The result is combined with wAND and wXOR parameters. To read the data bus without change, set wAND=0xFFFF and wXOR=0x0000.

For SC1x:

```
unsigned char hal_read_bus ( unsigned short adr,
                             unsigned short wAND,
                             unsigned short wXOR );
```

For SC1x3:

```
unsigned short hal_read_bus ( unsigned short adr,
                              unsigned short wAND,
                              unsigned short wXOR );
```

This function is **deprecated**! Use inport or inportb function from dos.h instead!

## Parameters

*adr*

I/O Address

*wAND*

Result mask

*wXOR*

Bit Inverter

## SC1x Return Value

8 Bit data in AX, AX = (databus & wAND) ^ wXOR

## SC1x3 Return Value

16 Bit data in AX, AX = (databus & wAND) ^ wXOR

## Comments

& = bit wise AND
^ = bit wise XOR

## See Also

**hal_readblock_bus** **hal_write_bus**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_write_bus

Write single byte to data bus. The provided parameters are combined as follows to form the output byte value:

```
output value = (data & wAND) ^ wXOR
```

To write the value to the address without modification, set wAND=0xFFFF and wXOR=0x0000.

For SC1x:

```
void hal_write_bus ( unsigned short adr, unsigned char value,
                     unsigned short wAND,unsigned short wXOR );
```

For SC1x3:

```
void hal_write_bus ( unsigned short adr, unsigned short value,
                     unsigned short wAND,unsigned short wXOR );
```

This function is **deprecated**! Use outport or outportb function from dos.h instead!

## Parameters

*adr*

I/O Address

*value*

Value to be output

*wAND*

Mask applied to output

*wXOR*

Output bit inverter

## Return Value

-- none --.

## Comments

& = bit wise AND
^ = bit wise XOR

## See Also

hal_writeblock_bus  hal_read_bus

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## hal_readblock_bus

Block read data bus.

```
void hal_readblock_bus ( unsigned short adr0, unsigned short adr1,
                         unsigned char far *buf, unsigned short cnt );
```

This function is **deprecated**! Use inport or inportb function from dos.h instead!

## Parameters

*adr0*

First address

*adr1*

Second address

*buf*

Output parameter: destination buffer

*cnt*

Number of bytes to read into `buf`

## Return Value

-- none --.

## Comments

IF `adr0` != `adr1` , this function will alternate reads between the two addresses until `cnt` bytes are read, starting at first address. Set `adr0` == `adr1` if you want to read from only a single address.

## See Also

hal_writeblock_bus  hal_read_bus

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_writeblock_bus

Block write data bus.

```
void hal_writeblock_bus ( unsigned short adr0, unsigned short adr1,
                          const unsigned char far *buf,
                          unsigned short cnt );
```

This function is **deprecated**! Use outport or outportb function from dos.h instead!

## Parameters

*adr0*

First address

*adr1*

Second address

*buf*

Input parameter: data source buffer

*cnt*

Number of bytes to output from `buf`

## Return Value

-- none --.

## Comments

IF `adr0` != `adr1` , this function will alternate between writes to the first and second address, starting at first address.

Set `adr0` == `adr1` if you want all data written to a single address.

## See Also

hal_readblock_bus  hal_write_bus

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_read_pios

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 = wInp value output from the **pfe_get_pio_state** function.

```
unsigned short hal_read_pios ( unsigned short wAND,
                               unsigned short wXOR );
```

## Parameters

*wAND*

mask

*wXOR*

bit inverter

## SC1x Return Value

(PIO[0..13] & wAND) ^ wXOR

## SC1x3 Return Value

(PIO[0..15] & wAND) ^ wXOR

## SC23 Return Value

(PIO[0..2, 9..13] & wAND) ^ wXOR

## Comments

& = bit wise AND
^ = bit wise XOR

## See Also

**hal_read_pio** **hal_write_pios** **pfe_get_pio_state**
**pfe_enable_pio**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_read_pios2

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` = `wInp` value output from the **pfe_get_pio_state2** function.

```
unsigned short hal_read_pios2 ( unsigned short wAND,
                                unsigned short wXOR );
```

## Parameters

*wAND*

mask

*wXOR*

bit inverter

## SC1x3 Return Value

(PIO[16..24, 26..31] & wAND) ^ wXOR

## SC23 Return Value

(PIO[18..23, 27..28, 31] & wAND) ^ wXOR

## Comments

& = bit wise AND
^ = bit wise XOR

## See Also

**hal_read_pio** **hal_write_pios2** **pfe_get_pio_state2**
**pfe_enable_pio**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_write_pios

Write to the programmable I/O pins. Before the value is written, the value is combined with the wAND and wXOR parameters as:

SC1x: PIO[0..13] = (data & wAND) ^ wXOR
SC1x3: PIO[0..15] = (data & wAND) ^ wXOR
SC23: PIO[0..2, 9..13] = (data & wAND) ^ wXOR

To write value 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.

```
void hal_write_pios ( unsigned short value,
                      unsigned short wAND,
                      unsigned short wXOR );
```

## Parameters

*value*

output data

*wAND*

mask

*wXOR*

bit inverter

## Return Value

-- none --.

## Comments

& = bit wise AND
^ = bit wise XOR

## See Also

hal_write_pio hal_read_pios pfe_enable_pio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V1.07 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_write_pios2

Write to the programmable I/O pins. Before the value is written, the value is combined with the wAND and wXOR parameters as:

SC1x3: PIO[16..24, 26..31] = (data & wAND) ^ wXOR
SC23: PIO[18..23, 27..28, 31] = (data & wAND) ^ wXOR

To write value 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.

```
void hal_write_pios2 ( unsigned short value,
                       unsigned short wAND,
                       unsigned short wXOR );
```

## Parameters

*value*

output data

*wAND*

mask

*wXOR*

bit inverter

## Return Value

-- none --.

## Comments

& = bit wise AND
^ = bit wise XOR

## See Also

hal_write_pio hal_read_pios2 pfe_enable_pio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_read_pio

Read specific I/O pin.

```
unsigned char hal_read_pio ( unsigned char pio );
```

## SC1x Parameters

*pio*
IPC@CHIP® PIO No. [0..13]

## SC1x3 Parameters

*pio*
IPC@CHIP® PIO No. [0..24, 26..31]

## SC23 Parameters

*pio*
IPC@CHIP® PIO No. [0..2, 9..13, 18..23, 27..28, 31]

## Return Value

0 PIO pin is low, 1 PIO pin is high

## Comments

Read specified programmable I/O pin.

## See Also

hal_write_pio  hal_read_pios  pfe_enable_pio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_write_pio

Write to specific I/O pin.

```
void hal_write_pio ( unsigned char pio, unsigned short value );
```

## Parameters

*value*

0 ==> set PIO to low
non-zero ==> set PIO to high

## SC1x Parameters

*pio*

IPC@CHIP® PIO No. [0..13]

## SC1x3 Parameters

*pio*

IPC@CHIP® PIO No. [0..24, 26..31]

## SC23 Parameters

*pio*

IPC@CHIP® PIO No. [0..2, 9..13, 18..23, 27..28, 31]

## Return Value

-- none --.

## Comments

Write to a specified programmable I/O pin. Only PIO pins that are defined as outputs can be written.

## See Also

hal_read_pio  hal_write_pios  pfe_enable_pio

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_install_isr

Install Interrupt Service Routine.

```
InterruptHandler hal_install_isr ( unsigned short irq,
                                   unsigned short count,
                                   InterruptHandler handler );
```

## Parameters

*count*

Number of interrupts generated before new user interrupt service routine is called.
count = 0 disables the user ISR (same as a NULL in handler)

*handler*

far pointer to user interrupt service routine
if pointer is NULL user ISR is disabled

## SC1x Parameters

*irq*

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) / DMA Interrupt Channel 0 (if DMA is used)
6 = INT6 (external) / DMA Interrupt 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 = *(*) DMA Interrupt Channel 2 (internal, not on SC12)*
14 = *(*) DMA Interrupt Channel 3 (internal, not on SC12)*
15 = NMI Powerfail Interrupt (internal/external)

(*) = used internally by @CHIP-RTOS, not available for user interrupt service functions

## SC1x3 Parameters

*irq*

HAL interrupt number from following list:

0 = *(*) Network controller 0 (internal)*
1 = INT1 (external)
2 = SPI controller (internal)
3 = INT3 (external) / *(*) USB controller (internal)*
4 = *(*) Serial port 2 (internal)*
5 = DMA Interrupt Channel 0
6 = DMA Interrupt Channel 1
7 = INT5 (external) / *(*) Serial port 3 (internal)*
8 = Timer0 (internal)
9 = Timer1 (internal)
10 = *(*) Timer 1ms (internal)*
11 = *(*) Serial port 0 (internal)*
12 = *(*) Serial port 1 (internal)*
13 = DMA Interrupt Channel 2
14 = DMA Interrupt Channel 3
15 = PFI Powerfail Interrupt (internal/external)
16 = *(*) CAN0 controller (internal) / (*) CAN1 controller (internal)*
17 = I2C controller (internal)
18 = Network controller 1 (internal)

(*) = used internally by @CHIP-RTOS, not available for user interrupt service functions

## SC23 Parameters

*irq*

HAL interrupt number from following list:

0 = *(*) Network controller 0 (internal)*
1 = INT1 (external)
2 = SPI controller (internal)
3 = INT3 (external) / *(*) USB controller (internal)*
4 = *(*) Serial port 2 (internal)*
5 = *(*) DMA Interrupt Channel 0*
6 = *(*) DMA Interrupt Channel 1*
7 = INT5 (external) / *(*) Serial port 3 (internal)*
8 = Timer0 (internal)
9 = Timer1 (internal)
10 = *(*) Timer 1ms (internal)*
11 = *(*) Serial port 0 (internal)*
12 = *(*) Serial port 1 (internal)*
13 = *(*) DMA Interrupt Channel 2*
14 = *(*) DMA Interrupt Channel 3*
15 = PFI Powerfail Interrupt (internal/external)
16 = *(*) CAN0 controller (internal) / (*) CAN1 controller (internal)*
17 = I2C controller (internal)
18 = *(*) Network controller 1 (internal)*

(*) = used internally by @CHIP-RTOS, not available for user interrupt service functions

## Return Value

Far pointer to old ISR handler

## 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 can be declared in either of the following forms:

```
void huge My_ISR(void) ;     // More efficient form
```

```
void interrupt My_ISR(void) ;     // Tolerated form
```

The more efficient `huge` procedures are recommended. The user ISR function must preserve only the DS and BP registers, so there is no requirement for the full register save/restore provided by the `interrupt` declarations.

Any required EOI signal is issued by the system ISR which calls your user ISR function. This EOI is issued after your function returns.

The `InterruptHandler` vector type is defined as:

```
typedef void huge (far *InterruptHandler)(void);
```

## See Also

**hal_install_rtx_isr** **pfe_enable_int** **RTX_Install_RTI_Hook**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_give_eoi

Give EOI to interrupt controller.

```
void hal_give_eoi ( unsigned char intr_no );
```

## Parameters

*intr_no*

Type (0=INT0 ... 4=INT4)

## Return Value

-- none --.

## Comments

Give End-Of-Interrupt for INT0-INT4.

When installing a interrupt service routine through HW API, it's not neccessay 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 **hal_install_isr** function. Especially when using cascaded mode of the interrupt controller with INT0/INTA, this function is needed for generating an EOI for INT0.

## See Also

**pfe_enable_int_cascade_mode**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | n/a | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_mask_int

Mask and Unmask an external Interrupt Request.

```
void hal_mask_int ( unsigned short irq, unsigned char mask );
```

## Parameters

*mask*

1: Mask IRQ
0: Unmask IRQ

## SC1x Parameters

*irq*

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) / DMA Interrupt Channel 0 (if DMA is used)
6 = INT6 (external) / DMA Interrupt 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 = DMA Interrupt Channel 2 (internal, not on SC12)
14 = DMA Interrupt Channel 3 (internal, not on SC12)
15 = NMI (Not maskable!)

## SC1x3/SC23 Parameters

*irq*

HAL interrupt number from following list:
0 = Network controller 0 (internal)
1 = INT1 (external) / Network controller 1 (internal)
2 = SPI controller (internal) / I2C controller (internal)
3 = INT3 (external) / USB controller (internal)
4 = Serial port 2 (internal)
5 = DMA Interrupt Channel 0
6 = DMA Interrupt Channel 1
7 = INT5 (external) / Serial port 3 (internal)

8 = Timer0 (internal)
9 = Timer1 (internal)
10 = Timer 1ms (internal)
11 = Serial port 0 (internal)
12 = Serial port 1 (internal)
13 = DMA Interrupt Channel 2
14 = DMA Interrupt Channel 3
15 = NMI (Not maskable)
16 = CAN0 controller (internal) / CAN1 controller (internal)

## Return Value

-- none --.

## Comments

**CAUTION:**
Masking any of the three timer interrupts will suspend the @CHIP-RTOS 1000 Hz real-time interrupt, essential for system operation. Consequently this mask period should be very brief, if used at all.

## SC1x Comments

Some interrupts share the same mask bit. If you mask one of them, the other interrupts which are assigned to the same bit are also masked. Here are the groups which are masked together:

Timer0, Timer1, Timer 1ms
DMA0, INT5
DMA1, INT6

## SC1x3/SC23 Comments

Some interrupts share the same mask bit. If you mask one of them, the other interrupts which are assigned to the same bit are also masked. Here are the groups which are masked together:

Timer0, Timer1, Timer 1ms
INT1 / Network controller 1
SPI controller / I2C controller
INT3 / USB controller
INT5 / Serial port 2
CAN0 controller / CAN1 controller

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

## hal_init_timer

Initialize timer settings

```
void hal_init_timer ( unsigned char timer,
                      unsigned short mode,
                      unsigned short divider );
```

## Parameters

*timer*

Timer
0=Timer0 / 1=Timer1

*mode*

Timer 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

*divider*

Clock divider (maximum count value)

## Return Value

-- none --.

## Comments

Initialize the timer settings of timer0 or timer1.

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_get_frequencies** function) or can also be configured to increment based on the TMRIN0 or TMRIN1 external signals (see PFE **pfe_enable_timer**). 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 **hal_set_duty_cycle_waveform** function).

Note that TMRIN pin and TMROUT pin can not be used at the same time.

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 **hal_install_isr** function to install your interrupt service routine.

If "use internal clock" is selected the associated TMRIN pin serves as a gate. A "high" on the TMRIN pin keeps the timer counting. A "low" holds the timer value.

## SC1x Comments

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).

## SC1x3/SC23 Comments

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.
If the clock divider value is set to 0000h, the timer will not count from 0000h to FFFFh like on SC12 and SC13.

## See Also

**hal_init_timer_ext** **hal_start_timer** **hal_read_timer_count**
**hal_write_timer_count**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

---

## Developer Notes

The timer output frequency is dependent on the internal processor clock.
For compatibility with future versions of @Chip, please use the HAL **hal_get_frequencies** to compute the correct clock divider value.

Available examples

1. TimerIn example, timerin.c
2. TimerOut example, timerout.c

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

## hal_init_timer_ext

Initialize timer settings Extended

```
void hal_init_timer_ext ( unsigned char timer,
                          unsigned short mode );
```

## Parameters

*timer*

Timer
0=Timer0 / 1=Timer1

*mode*

Timer mode:
Bit 0..2: must be 0
Bit 3: 0=disable prescale t2 / 1=enable prescale t2
Bit 4: 0=disable retrigger / 1=enable retrigger
Bit 5..15: must be 0

## Return Value

-- none --.

## Comments

This function must be called after the Initialize Timer Settings **hal_init_timer** function call!

The mode control bits have the following affect:

**Prescale T2:** If the Prescale feature is enabled, the timer (specified by argument timer) will use the Timer2 output for its time base. If the **RTI rate** is not changed by the user it will provide a 1000 Hz timer clock rate. (Timer2 is used internally by the @CHIP-RTOS as the Real-Time Interrupt (RTI), which defaults to a one millisecond interval timer.) If the Prescale feature is disabled, then the value returned from the **hal_get_frequencies** function indicates the timer's time base.

## SC1x Comments

**Retrigger:** If the retrigger setting is enabled, then a 0 to 1 edge transition on TMRIN0 or TMRIN1 resets the respective counter to zero. When retrigger setting is disabled, a High input on TMRIN0 or TMRIN1 enables counting and a Low input holds the current timer value.

## SC1x3/SC23 Comments

**Retrigger:** If the retrigger setting is enabled, then a 0 to 1 edge transition on TMRIN0 or TMRIN1 resets the respective counter to zero. If the input remains High, the timer starts counting on the following clock transitions. If the input goes again Low, the timer stops counting. When retrigger setting is disabled, a High input on TMRIN0 or TMRIN1 enables counting and a Low input holds the current timer value.

## See Also

**hal_init_timer** **hal_start_timer** **hal_read_timer_count**
**hal_write_timer_count**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_start_timer

Start timer

```
void hal_start_timer ( unsigned char timer );
```

## Parameters

*timer*

Which timer:
0 = Timer0 / 1 = Timer1

## Return Value

-- none --.

## Comments

Enable the specified timer to up count.

## See Also

hal_init_timer  hal_stop_timer

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## hal_stop_timer

Stop timer

```
void hal_stop_timer ( unsigned char timer );
```

## Parameters

*timer*

Which timer:
0 = Timer0 / 1 = Timer1

## Return Value

-- none --.

## Comments

Stop the specified timer's counting.

## See Also

hal_init_timer  hal_start_timer

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## hal_read_timer_count

Read timer count

```
unsigned short hal_read_timer_count ( unsigned char timer,
                                      unsigned short far * maxcount );
```

## Parameters

*timer*

Which timer:
0 = Timer0 / 1 = Timer1

*maxcount*

Optional output parameter.
If this pointer is not NULL, then a Boolean is output:
0 ->Maximum count not reached
1 ->Maximum count reached
Set to NULL if not interested in maximum count.

## Return Value

Current count of the associated timer

## Comments

Read the timer count.

Return value contains the current count of the associated timer. The count is incremented by the processor internal clock (see HAL **hal_get_frequencies** function), unless the timer is configured for external clocking (then it is clocked by the TMRIN0 and TMRIN1 inputs, see **pfe_enable_timer** function).

## See Also

**hal_init_timer** **hal_stop_timer** **hal_write_timer_count**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_write_timer_count

Write timer count

```
void hal_write_timer_count ( unsigned char timer,
                             unsigned short counter );
```

## Parameters

*timer*

Which timer:
0 = Timer0 / 1 = Timer1

*counter*

Value to write to 16 bit counter

## Return Value

-- none --.

## Comments

Preset the specified timer's count register to provided value.

The timer count can be written at any time, regardless of whether the corresponding timer is running.

## See Also

hal_init_timer  hal_stop_timer  hal_read_timer_count

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_set_duty_cycle_waveform

Set timer duty cycle waveform

```
void hal_set_duty_cycle_waveform ( unsigned char timer,
                                   short mode,
                           unsigned short clock_divider );
```

## Parameters

*timer*

Which timer:
0 = Timer0 / 1 = Timer1

*mode*

0=disable duty cycle
1=enable duty cycle

*clock_divider*

Alternate clock divider (if mode = 1)

## Return Value

-- none --.

## Comments

Set the duty cycle waveform of specified timer.

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 `clock_divider` that is the same value as was used for the main *clock divider* value set in the Timer Initialization **hal_init_timer** 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.

## See Also

**hal_init_timer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_set_watchdog_mode

Get/Set watchdog mode.

```
unsigned char hal_set_watchdog_mode ( unsigned char mode );
```

## Parameters

*mode*

0 = only get mode
2 = Watchdog will be triggered by user program
3 = Watchdog will be triggered by @CHIP-RTOS (default)

## Return Value

Watchdog mode:
2 = Watchdog will be triggered by user program
3 = Watchdog will be triggered by @CHIP-RTOS (default)

## Comments

Get or set the watchdog mode.

In @CHIP-RTOS mode, the @CHIP-RTOS 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.

## SC1x Comments

The watchdog timeout period on SC11/SC12/SC13 is 800 ms.
If you select the *User Program* mode, you must call the HAL **hal_refresh_watchdog** function at least every 800 ms to prevent the watchdog from resetting the system.

## SC1x3/SC23 Comments

The watchdog timeout period on SC1x3/SC23 is 699 ms.
If you select the *User Program* mode, you must call the HAL **hal_refresh_watchdog** function at least every 699 ms to prevent the watchdog from resetting the system.

## See Also

**hal_refresh_watchdog**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_refresh_watchdog

Refresh watchdog.

```
void hal_refresh_watchdog ( void );
```

## Parameters

-- none --

## Return Value

-- none --.

## SC1x Comments

Strobe the hardware watchdog to reset its 800 ms timeout period.

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.

## SC1x3/SC23 Comments

Strobe the hardware watchdog to reset its 699 ms timeout period.

If the watchdog is in *User Program* mode, this function must be called at least every 699 ms to prevent a CPU reset due to watchdog timeout.

## See Also

hal_set_watchdog_mode

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## hal_start_dma

Start DMA mode.

```
int hal_start_dma ( unsigned char channel,
                    const Dma_Info * dmaInfo );
```

## Parameters

*dmaInfo*

Pointer to **Dma_Info** data structure containing DMA setup information.

## SC1x Parameters

*channel*
DMA channel:
0 = DRQ0
1 = DRQ1

## SC1x3 Parameters

*channel*
DMA channel:
0 = DRQ0
1 = DRQ1
2 = DRQ2
3 = DRQ3

## Return Value

0: Success
-1: Invalid DMA channel
-2: DMA channel used for serial interface

## Comments

This function configures the specified DMA channel based on the contents of the provided `dmaInfo` **structure**. Then the DMA channel is activated.

## See Also

hal_stop_dma hal_get_dma_info

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_stop_dma

Stop DMA transfer.

```
int hal_stop_dma ( unsigned char channel );
```

## SC1x Parameters

*channel*
DMA channel:
0 = DRQ0
1 = DRQ1

## SC1x3 Parameters

*channel*
DMA channel:
0 = DRQ0
1 = DRQ1
2 = DRQ2
3 = DRQ3

## Return Value

0: Success
-1: Invalid DMA channel
-2: DMA channel used for serial interface

## Comments

Stops a DMA transfer (disables the DMA controller). The transfer could be continued by first calling **hal_get_dma_info** to gather the current DMA state and then passing this state information to **hal_start_dma** to resume the transfer.

## See Also

**hal_start_dma** **hal_get_dma_info**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_get_dma_info

Get DMA info.

```
int hal_get_dma_info ( unsigned char channel,
                       Dma_Info * dmaInfo );
```

## Parameters

*dmaInfo*

Output Parameter: Pointer to **Dma_Info** data structure which will be set by this API with DMA state information.

## SC1x Parameters

*channel*

DMA channel:

0 = DRQ0
1 = DRQ1

## SC1x3 Parameters

*channel*

DMA channel:

0 = DRQ0
1 = DRQ1
2 = DRQ2
3 = DRQ3

## Return Value

Existing DMA control **register** bit field.

## Comments

The DMA is momentarily disabled during reading of status, and then restored to the enable state it was at.

## See Also

**hal_start_dma** **hal_stop_dma**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | n/a |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_init_non_volatile_data

Initialize/Restore non-volatile data.

```
int hal_init_non_volatile_data ( Remop far * remop );
```

## Parameters

*remop*

Pointer to **Remop** data structure.

## Return Value

Success: AX = 0
Failure: AX < 0, Could not create file

## Comments

This function initializes and restore non-volatile data. Use it to tell the @CHIP-RTOS where your variables are located, which should be saved. The function will also reload their saved values, if available.

Call this function at the beginning of your program.

The **Remop** structure reference by `remop` must be static. (This function does not make a copy of the structure's content.)

## SC1x Comments

The non-volatile (remanent) data is stored in `A:\rema.bin` file.

## SC1x3/SC23 Comments

The non-volatile (remanent) data is stored in a reserved flash memory area. The maximum size is approx. 8 KB.

## See Also

**hal_save_non_volatile_data** **hal_get_reboot_reason**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_save_non_volatile_data

Save non-volatile data.

```
void hal_save_non_volatile_data ( void );
```

## Parameters

-- none --

## Return Value

-- none --.

## Comments

Call this function on exit from your program and in your PFI (Power Fail Interrupt) handler. Your hardware around the IPC@CHIP® must support the PFAIL signal, so that the IPC@CHIP® can generate an PFI/NMI sufficiently in advance of power loss to the CPU.

## SC1x Comments

**Reminder** : The DK40 does not support the PFAIL signal.

## See Also

hal_init_non_volatile_data

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_get_reboot_reason

Get reboot reason.

```
int hal_get_reboot_reason ( void );
```

## Parameters

-- none --

## Return Value

reason:

0: REBOOT_UNKNOWN
3: REBOOT_WATCHDOG
4: REBOOT_POWER_FAIL

The constants referred to here are defined in <HWAPI.H> header file.

## Comments

When called from a user program, this function only returns valid results if the init/restore **hal_init_non_volatile_data** function was called following the reboot.

This function may also be used from within an NMI **callback** to access the cause of the current shut down interrupt. In this case the reading will be either `REBOOT_UNKNOWN` or `REBOOT_POWER_FAIL`.

## See Also

**hal_init_non_volatile_data** **hal_save_non_volatile_data**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_install_pfaildone_callback

Install powerfaildone callback.

```
void hal_install_pfaildone_callback (InterruptHandler pfaildone_callback);
```

## Parameters

*pfaildone_callback*

Pointer to callback function.

## Return Value

none

## Comments

Install a callback for the event "Powerfail is Done". The callback is executed after all Powerfail actions are finished. This means that the user non-volatile data has been written and the file sytem closed.

The user can toggle a PIO inside this callback and measure the time from Powerfail (PFI/NMI) to this event.

The time to close the file system depends on the number of open files and on which drive these files are located (speed of the drive).

## See Also

hal_init_non_volatile_data  hal_save_non_volatile_data

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.21 | V1.21 | V1.21 | V1.10 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## hal_get_frequencies

Get the specified system frequency.

```
unsigned long hal_get_frequencies (
                          HW_FREQ_SELECTOR freq_select );
```

## SC1x Parameters

*freq_select*

Enumerator, which frequency to get:

GET_FRQ_PROCESSOR = 0: Return processor frequency
GET_FRQ_TIMER = 1: Return maximum TMROUT square wave frequency
GET_FRQ_MAX_BAUD = 2: Return maximum baud rate
GET_FRQ_PWD = 3: PWD timer frequency
GET_FRQ_TIMERBASE = 3: Return timer base frequency

## SC1x3/SC23 Parameters

*freq_select*

Enumerator, which frequency to get:

GET_FRQ_PROCESSOR = 0: Return processor frequency
GET_FRQ_TIMER = 1: Return maximum TMROUT square wave frequency
GET_FRQ_MAX_BAUD = 2: Return maximum baud rate
GET_FRQ_TIMERBASE = 3: Return timer base frequency

## Return Value

frequency [Hz].

## Comments

Use the timer base frequency to compute the correct timer clock divider value, where:

```
Timer frequency = timer base frequency / clock divider
```

Use the maximum TMROUT sqare wave frequency to compute the correct timer clock divider value for the TMROUT, where:

```
TMROUT frequency = maximum TMROUT sqare wave frequency / clock divider
```

Use the maximum baud rate to select the correct baud rate divisor for the specific processor. The `baud_divider` argument to Fossil *Extended line control initialization* **fossil_set_extctrl** function can be calculated based on the following equation.

```
baud_divider = maximum baud rate / desired baud rate
```

## See Also

**hal_init_timer** **fossil_set_extctrl**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS C Library - I2C Bus API

---

Header File: `#include <I2C_API.H>`

**Control**

**Transmit**

**Receive**

## Control Functions

| | |
|---|---|
| I2C_init | I2C Master: Initialize the I2C Bus |
| I2C_scan | I2C Master: Scan for I2C slave devices |
| I2C_scan_ext | I2C Master: Scan for I2C slave devices (extended address) |
| I2C_release | I2C Master: Release I2C bus |
| I2C_restart | I2C Master: Restart I2C |
| I2C_select_clock_pin | I2C Master: Select I2C Clock Pin |
| I2C_select_data_pin | I2C Master: Select I2C Data Pin |
| I2C_set_speed | I2C Master/Slave: Select I2C bus speed |
| I2C_reset | I2C Master/Slave: Reset I2C bus controller |
| I2C_set_timeout | I2C Master: Set I2C bus timeout |

## Transmit Functions

| | |
|---|---|
| I2C_transmit_char | I2C Master: Send a single character |
| I2C_transmit_block | I2C Master: Send a block of characters |
| I2C_transmit_char_ext | I2C Master: Send a single character (extended address) |
| I2C_transmit_block_ext | I2C Master: Send a block of characters (extended address) |

## Receive Functions

| | |
|---|---|
| I2C_receive_char | I2C Master: Receive a single character |
| I2C_receive_block | I2C Master: Receive block of characters |

| | |
|---|---|
| **I2C_receive_char_ext** | I2C Master: Receive a single character (extended address) |
| **I2C_receive_block_ext** | I2C Master: Receive block of characters (extended address) |

## Slave Functions

| | |
|---|---|
| **I2C_slave_address** | I2C Slave: Set I2C bus slave address |
| **I2C_slave_getstate** | I2C Slave: Get I2C bus state |
| **I2C_slave_sendack** | I2C Slave: Send a I2C bus acknowledge |
| **I2C_slave_readbyte** | I2C Slave: Read byte from I2C bus |
| **I2C_slave_sendbyte** | I2C Slave: Send byte to I2C bus |
| **I2C_slave_idle** | I2C Slave: Set the I2C internal state back to idle |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_init

Initialize the I2C Bus

```
void I2C_init ( void );
```

## Parameters

-- none --

## Return Value

-- none --

## Comments

This function configures two of the programmable I/O (PIO) pins for usage as I2C bus data and clock signals.

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.

## SC1x3/SC23 Comments

On SC1x3/SC23 the default is to use the hardware I2C interface on the dedicated pins. If the user calls functions 8eh and 8fh to select other PIOs for the bus, the I2C bus will be software emulated on the selected PIO pins.

If the hardware interface is used, the calling task will go to sleep, until the I2C interrupt issues a wakeup. Therefore the user should not make own wakeup calls to the I2C API calling task, while sleeping inside I2C functions.

## See Also

**I2C_select_clock_pin** **I2C_select_data_pin**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

PFE: **Enable** Programmable I/O Pins

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_scan

Scan for I2C slave devices

```
unsigned char I2C_scan ( unsigned char start_addr,
                         unsigned char end_addr );
```

## Parameters

*start_addr*

First slave address (even address, LSB=0)

*end_addr*

Last slave address (even address, LSB=0)

## Return Value

0 => no slave found
Non-zero => address of the first slaved device found

## 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.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## I2C_scan_ext

Scan for I2C slave devices (extended address)

```
unsigned int I2C_scan_ext ( unsigned int start_addr,
                            unsigned int end_addr );
```

## Parameters

*start_addr*

First slave address (even address, LSB=0)

*end_addr*

Last slave address (even address, LSB=0)

## Return Value

0 => no slave found
Non-zero => address of the first slaved device found

## 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.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - I2C Bus API

## I2C_release

Release I2C Bus

```
void I2C_release ( void );
```

## Parameters

-- none --

## Return Value

-- none --

## Comments

This function provides the I2C bus stop condition: a data low to high transition during clock high. Both clock and data output pins are left tri-stated.

## See Also

I2C_restart

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## I2C_restart

Restart I2C Bus

```
void I2C_restart ( void );
```

## Parameters

-- none --

## Return Value

-- none --

## Comments

This function provides the I2C bus start condition: a data high to low transition during clock high.

## See Also

I2C_release

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## I2C_select_clock_pin

Select I2C Clock Pin

```
void I2C_select_clock_pin ( unsigned char pio_no );
```

## SC1x Parameters

*pio_no*
PIO pin number, [0..13]

## SC1x3 Parameters

*pio_no*
PIO pin number, [0..24, 26..31]

## SC23 Parameters

*pio_no*
PIO pin number, [0..1, 9..11, 13, 18..23, 27..28]

## Return Value

-- none --

## SC1x Comments

The default I2C clock pin is PIO 0.

To change the I2CCLK pin this function must be called before the **I2C_init** function is called.

## SC1x3/SC23 Comments

To use the I2C hardware interface do not call the functions "Select I2C Clock Pin" and "Select I2C Data Pin" or call them both with PIO pin number 0.

## See Also

**I2C_select_data_pin** **I2C_init**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_select_data_pin

Select I2C Data Pin

```
void I2C_select_data_pin ( unsigned char pio_no );
```

## SC1x Parameters

*pio_no*
PIO pin number, [0..13]

## SC1x3 Parameters

*pio_no*
PIO pin number, [0..24, 26..31]

## SC23 Parameters

*pio_no*
PIO pin number, [0..1, 9..11, 13, 18..23, 27..28]

## Return Value

-- none --

## SC1x Comments

The default I2C data pin is PIO 1.

To change the I2CDAT pin this function must be called before the **I2C_init** function is called.

## SC1x3/SC23 Comments

To use the I2C hardware interface do not call the functions "Select I2C Clock Pin" and "Select I2C Data Pin" or call them both with PIO pin number 0.

## See Also

**I2C_select_clock_pin** **I2C_init**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_set_speed

Select I2C frequency

```
void I2C_set_speed ( unsigned char n, unsigned char m );
```

## Parameters

*n*

parameter n (allowed value range 0-7)

*m*

parameter m (allowed value range 0-15)

## Return Value

-- none --

## Comments

This function sets the I2C bus speed, if the I2C hardware interface is used.

FREQsample = CPU clock / 2^N
FREQoscl = CPU clock / ((2^N) * (M+1) * 10)

FREQoscl is the frequency of the I²C clock line when the I²C controller is in master mode. FREQsample is the frequency at which the I²C bus is sampled. This frequency should be at least 10 times faster than the fastest master on the bus, to ensure that START and STOP conditions are always detected.

Default is N=4, M=5, which sets FREQsample to 6 MHz and FREQoscl to 100 KHz at 96 MHz CPU frequency.

## See Also

I2C_init

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - I2C Bus API

## I2C_reset

Reset I2C bus hardware controller

```
void I2C_reset ( void );
```

## Return Value

-- none --

## Comments

This function resets the I2C bus hardware controller.

## See Also

I2C_init

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## I2C_set_timeout

Set I2C bus timeout

```
void I2C_set_timeout ( unsigned int value );
```

## Parameters

*value*

Timeout value.

If the I2C hardware interface is used, this value has a resolution in milliseconds. This means that the internal I2C functions will wait for BX milliseconds if the I2C clock signal is released by the I2C slave.

If the software I2C emulation is used, this value is interpreted as a cycle count. This means that the internal I2C functions check BX times if the I2C clock signal is released by the I2C slave.

## Return Value

-- none --

## Comments

If the I2C slave uses clock stretching, this timeout specifies the maximum waiting time, before the access is aborted.

The default timeout is:
SC12: 50 cycles
SC12B: 100 cycles
SC13/SC11: 250 cycles
SC1x3/SC23: 3000 cycles / 50 ms

## See Also

I2C_init

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_transmit_char

Send a single character

```
int I2C_transmit_char ( unsigned char slave,
                        char char_out );
```

## Parameters

*slave*

7 bit slave address in B7-B1, B0=Don't care

*char_out*

Byte to transmit.

## Return Value

0: Success
8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Return Value

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read
(change from master to slave)
5: Arbitration lost, own slave address received, master wants to write
(change from master to slave)
6: Arbitration lost
7: Bus error

## Comments

The IPC@CHIP® is the I2C bus master.

This function will automatically generated a I2C Start or Restart condition.

## See Also

I2C_send_block  I2C_receive_char

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_transmit_block

Send block of data on I2C bus

```
int I2C_transmit_block ( unsigned char slave,
                         const char far *buffer,
                         int length );
```

## Parameters

*slave*

7 bit slave address in B7..B1; B0 n/a

*buffer*

Pointer to data source

*length*

Number of bytes to send from buffer

## Return Value

result - 0 = Success
non-zero indicates error:
8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Return Value

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read
(change from master to slave)
5: Arbitration lost, own slave address received, master wants to write
(change from master to slave)
6: Arbitration lost
7: Bus error

## Comments

This function will dwell until 'length' bytes from the buffer are transmitted or until an error occurs.

This function will automatically generated a I2C Start or Restart condition.

## See Also

I2C_transmit_char  I2C_receive_block

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---

## I2C_transmit_char_ext

Send a single character (extended address)

```
int I2C_transmit_char_ext ( unsigned int slave,
                            char char_out );
```

## Parameters

*slave*

10 bit slave address in B10-B1, B0=Don't care

*char_out*

Byte to transmit.

## Return Value

0: Success
8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Return Value

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read
(change from master to slave)
5: Arbitration lost, own slave address received, master wants to write
(change from master to slave)
6: Arbitration lost
7: Bus error

## Comments

The IPC@CHIP® is the I2C bus master.

This function will automatically generated a I2C Start or Restart condition.

## See Also

I2C_send_block_ext  I2C_receive_char_ext

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_transmit_block_ext

Send block of data on I2C bus (extended address)

```
int I2C_transmit_block_ext ( unsigned int slave,
                             const char far *buffer,
                             int length );
```

## Parameters

*slave*

10 bit slave address in B10..B1; B0 n/a

*buffer*

Pointer to data source

*length*

Number of bytes to send from buffer

## Return Value

result - 0 = Success
non-zero indicates error:
8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Return Value

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read
(change from master to slave)
5: Arbitration lost, own slave address received, master wants to write
(change from master to slave)
6: Arbitration lost
7: Bus error

## Comments

This function will dwell until 'length' bytes from the buffer are transmitted or until an error occurs.

This function will automatically generated a I2C Start or Restart condition.

## See Also

I2C_transmit_char_ext  I2C_receive_block_ext

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_receive_char

Receive a single character

```
int I2C_receive_char ( unsigned char slave,
                       char far *char_in,
                       unsigned char not_last );
```

## Parameters

*slave*

7 bit slave address in B7-B1, B0=Don't care

*char_in*

Pointer to one character buffer provided by caller where received character will be placed.

*not_last*

= 0 for last char to be received, otherwise non-zero.

## Return Value

0: Success, received char is output to 'char_in' location.
8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Return Value

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read
(change from master to slave)
5: Arbitration lost, own slave address received, master wants to write
(change from master to slave)
6: Arbitration lost
7: Bus error

## Comments

The IPC@CHIP® is the I2C bus master.

This function will automatically generated a I2C Start or Restart condition.

## See Also

I2C_receive_block  I2C_transmit_char

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_receive_block

Receive block of data on I2C bus

```
int I2C_receive_block ( unsigned char slave,
                        char far *buffer,
                        int length );
```

## Parameters

*slave*

7 bit slave address in B7..B1; B0 n/a

*buffer*

Pointer to where data will be placed

*length*

Number of bytes to receive

## Return Value

result - 0 = Success
non-zero indicates error:
8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Return Value

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read
(change from master to slave)
5: Arbitration lost, own slave address received, master wants to write
(change from master to slave)
6: Arbitration lost
7: Bus error

## Comments

This function will dwell until either 'length' bytes are received and stored in the buffer, or until an error occurs.

This function will automatically generated a I2C Start or Restart condition.

## See Also

I2C_receive_char  I2C_transmit_block

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## I2C_receive_char_ext

Receive a single character (extended address)

```
int I2C_receive_char_ext ( unsigned int slave,
                           char far *char_in,
                           unsigned char not_last );
```

## Parameters

*slave*

10 bit slave address in B10-B1, B0=Don't care

*char_in*

Pointer to one character buffer provided by caller where received character will be placed.

*not_last*

= 0 for last char to be received, otherwise non-zero.

## Return Value

0: Success, received char is output to 'char_in' location.
8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Return Value

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read
(change from master to slave)
5: Arbitration lost, own slave address received, master wants to write
(change from master to slave)
6: Arbitration lost
7: Bus error

## Comments

The IPC@CHIP® is the I2C bus master.

This function will automatically generated a I2C Start or Restart condition.

## See Also

I2C_receive_block_ext  I2C_transmit_char_ext

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_receive_block_ext

Receive block of data on I2C bus (extended address)

```
int I2C_receive_block_ext ( unsigned int slave,
                            char far *buffer,
                            int length );
```

## Parameters

*slave*

10 bit slave address in B10..B1; B0 n/a

*buffer*

Pointer to where data will be placed

*length*

Number of bytes to receive

## Return Value

result - 0 = Success
non-zero indicates error:
8: Timeout
9: Slave faulty or not available

## SC1x3/SC23 Return Value

On SC1x3/SC23 these additional return values are possible if the hardware I2C interface is used:
3: Arbitration lost, general call address received
4: Arbitration lost, own slave address received, master wants to read
(change from master to slave)
5: Arbitration lost, own slave address received, master wants to write
(change from master to slave)
6: Arbitration lost
7: Bus error

## Comments

This function will dwell until either 'length' bytes are received and stored in the buffer, or until an error occurs.

This function will automatically generated a I2C Start or Restart condition.

## See Also

I2C_receive_char_ext  I2C_transmit_block_ext

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## I2C_slave_address

Set the I2C bus slave address

```
void I2C_slave_address ( unsigned int addr, unsigned char gencalladdr );
```

## Parameters

*addr*

Slave address:
Bit0: don't care
Bit7-1: 7 bit slave address or
Bit10-1: 10 bit slave address

*gencalladdr*

0: don't listen to general call address
1: listen to general call address

## Return Value

-- none --

## Comments

This function initializes the I2C bus slave interface. To use the I2C slave interface the user has to install an interrupt service routine for the I2C interrupt. An interrupt occurs when the I2C state changes. The user should than read the I2C state with function I2C_slave_getstate.

The I2C bus intialize function must be called previously. The Slave mode works only if the I2C hardware interface in used.

## See Also

I2C_init

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_slave_getstate

Get the I2C bus state

```
unsigned int I2C_slave_getstate ( void );
```

## Return Value

I2C bus state:
3: General call address received
4: Own slave address received, master wants to read
5: Own slave address received, master wants to write
6: Arbitration lost
7: Bus error
10: Data byte received from master
11: Data byte transmitted to master, master wants next byte
12: Last data byte transmitted to master
13: STOP or repeated START condition received

## Comments

This function reads the current I2C bus state and should be used inside the I2C interrupt handler, if the I2C slave interface is used.

## See Also

I2C_slave_address

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|

V2.01

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_slave_sendack

Send a I2C bus acknowledge

```
void I2C_slave_sendack ( unsigned char ack );
```

## Parameters

*ack*

0: Send a non acknowledge
1: Send acknowledge

## Return Value

-- none --

## Comments

This function should be used inside the I2C interrupt handler, if the I2C slave interface is used and the return state signals that the master wants to write a byte.

## See Also

I2C_slave_getstate

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_slave_readbyte

Read a byte from I2C bus as slave

```
unsigned char I2C_slave_readbyte ( void );
```

## Return Value

The read byte

## Comments

This function should be used inside the I2C interrupt handler, if the I2C slave interface is used and the return state signals that the master has written a byte, which can now be read out.

## See Also

I2C_slave_sendbyte

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

# @CHIP-RTOS C Library - I2C Bus API

## I2C_slave_sendbyte

Send a byte to I2C bus as slave

```
void I2C_slave_sendbyte ( unsigned char data, unsigend char next );
```

## Parameters

*data*

Byte to write

*next*

1: Signal that more bytes are available
0: Signal that this is the last byte that can be transfered

## Return Value

-- none --

## Comments

This function should be used inside the I2C interrupt handler, if the I2C slave interface is used and the return state signals that the master wants to read a byte.

## See Also

I2C_slave_readbyte

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## I2C_slave_idle

Set the I2C internal state back to idle

```
void I2C_slave_idle ( void );
```

## Return Value

-- none --

## Comments

This function should be used inside the I2C interrupt handler, if the I2C slave interface is used and the return state signals that the last byte was transmitted to the master.

## See Also

I2C_slave_getstate

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This API List
List of C Libraries
@CHIP-RTOS Main Index




# @CHIP-RTOS C Library - Packet Driver API

Header File: `#include <PKTDRV.H>`
Communication Configuration

## Communication Functions

| | |
|---|---|
| Set_Access_Type | Install an Access Handler |
| Set_WildcardAccess_Type | Install an Wildcard Access handler |
| Release_Type | Uninstall an Access Handler |
| Send_Packet | Send an Ethernet Packet |

## CONFIG Config Functions

| | |
|---|---|
| PktDriver_Installed | Is the Packet Driver available |
| Get_Driver_Info | Get Driver Info |
| Set_Receive_Mode | Set the receive mode of the Ethernet device |
| Get_Receive_Mode | Get the receive mode of the Ethernet device |
| Set_Multicast | Set or Delete an Ethernet Multicast MAC Address |
| Get_Eth_Address | Get the @CHIPs Ethernet MAC Address |

List of C Libraries
@CHIP-RTOS Main Index

End of document

## Set_Access_Type

Installs an access Handler for Ethernet Packets.

```
unsigned int Set_Access_Type ( const Driver_Info *drv_info,
                               const void far * Receiver,
                               const int *pkt_type,
                               int *handle );
```

## Parameters

*drv_info*

Pointer to struct returned from **Get_Driver_Info**. (This parameter is not used and may be left NULL.)

*Receiver*

Pointer to receiver handler function

*pkt_type*

Pointer to packet type (in network byte order, e.g. 0x0008 for IP). The value referenced here must be persistent due to that the @Chip-RTOS will reference this value during operation. (No copy is made by this API.)

*handle*

Output Parameter: Pointer where handle should be stored. This handle will be needed for the **Release_Type** API.

## Return Value

0 on Success
else error code:

```
5    BAD_TYPE      - Type not allowed 9    NO_SPACE      - Limit of 5
accesses reached
10    TYPE_INUSE    - The type is already accessed
```

## 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 RTOS variants 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, the received Ethernet data is now available in your buffer. The format of this received data fits the **Eth_Packet** type defined in the pktdrv.h header file.

  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:** Due to that the @Chip-RTOS 80x86 type processor is a little endian machine, exchange the bytes for the packet type e.g. use 0x0008 for the IP type instead of 0x0800. This way round, the "network byte order" requirement on this packet type is satisfied.

## See Also

**Get_Driver_Info** **Set_WildcardAccess_Type** **Release_Type**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Set_WildcardAccess_Type

Installs an wildcard packet access Handler for Ethernet Packets.

```
unsigned int Set_WildcardAccess_Type (
                    const Driver_Info *drv_info,
                    const void far * Receiver,
                    const int *pkt_type,
                    int *handle ) ;
```

## Parameters

*drv_info*

Pointer to struct returned from **Get_Driver_Info**. (This parameter is not used and may be left NULL.)

*Receiver*

Pointer to receiver handler function

*pkt_type*

Pointer to packet type (e.g. 0xFFFF for accepting every Ethernet packet). This referenced value must be persistent.

*handle*

Output Parameter: Pointer where handle should be stored. This handle will be needed for the **Release_Type** API.

## Return Value

0 on Success
else packet driver API is not supported by this @Chip-RTOS version.

## Comments

Only one wildcard could be installed. Installing a new type overwrites the previous.

If a packet type of 0xFFFF is installed, any incoming packet will be accepted.

A wild card handler is removed by installing a wildcard with the packet type 0x0000.

The "normal" handlers installed with **Set_Access_Type** API are not overwritten by installing a wildcard. Refer to the **Set_Access_Type** API description for explanation on how the installed callback function is

used.

## See Also

Set_Access_Type  Get_Driver_Info  Release_Type

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Packet Driver API

## Release_Type

Uninstalls an access Handler

```
unsigned int Release_Type( unsigned int handle );
```

## Parameters

*handle*

Handle of access handler which should be uninstalled (returned from Set_Access_Type).

## Return Value

0 on Success
else error code:

```
1    BAD_HANDLE    - handle not recognized
```

## See Also

Set_Access_Type

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## Send_Packet

Sends an Ethernet Packet

```
unsigned char Send_Packet ( const unsigned char *buffer,
                            unsigned int len );
```

## Parameters

*buffer*

Pointer to packet buffer

*len*

Length of packet buffer

## Return Value

0 on Success
else error code:

```
12    CANT_SEND    - Send failed
```

## Comments

The format of the outgoing packet is described by the **Eth_Packet** type defined in the pktdrv.h header file. The user must set the source and destination IP addresses and packet type in this structure.

## See Also

**Get_Eth_Address** **Set_Access_Type**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

# @CHIP-RTOS C Library - Packet Driver API

## PktDriver_Installed

Checks if the packet driver is available

```
unsigned char PktDriver_Installed ( void );
```

## Parameters

-- none --

## Return Value

1: installed
0: not installed

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Get_Driver_Info

Get information about the packet driver

```
unsigned char Get_Driver_Info( Driver_Info *drv_info ) ;
```

## Parameters

*drv_info*

Pointer to Driver_Info struct, where driver information should be stored

## Return Value

0

## See Also

Set_Access_Type

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## Set_Receive_Mode

Set receive mode of the Ethernet device

```
unsigned char Set_Receive_Mode( unsigned int mode,
                                unsigned char *error );
```

## Parameters

*mode*

receive mode

*error*

Output Parameter: Pointer where error code should be stored. Set to zero on success, else 11 (= BAD_COMMAND) on failure.

## Return Value

0 on Success
-1 invalid mode

## Comments

Settable modes:

```
RCV_BCAST //accept packets with own mac address or broadcast
RCV_ALL    //accept all incoming packets
```

## See Also

Get_Receive_Mode  Set_Multicast

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **n/a** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Get_Receive_Mode

Get the current receive mode of the Ethernet device

```
unsigned char Get_Receive_Mode( unsigned char *error );
```

## Parameters

*error*

Output Parameter: Pointer where error code should be stored. Set to zero on success. (No failure is expected here if @Chip-RTOS supports the packet driver API.)

## Return Value

0 on Success
-1 API failed

## Comments

Possible Receive modes:

```
RCV_BCAST         //accept own address and broadcasts
RCV_BCAST_MCAST //accept broadcasts and installed multicasts
RCV_ALL         //all incoming packets are accepted
```

## See Also

Set_Receive_Mode  Set_Multicast

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This API List

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Set_Multicast

Set Ethernet multicast Address

```
unsigned int Set_Multicast(const unsigned char *mac_addr,
                           int set ) ;
```

## Parameters

*mac_addr*

Pointer to multicast mac address buffer (6 bytes)

*set*

1 to set, 0 to delete

## Return Value

0 on Success
else packet driver API is not supported by this @Chip-RTOS version.

## See Also

Get_Receive_Mode  Set_Receive_Mode

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## Get_Eth_Address

Get the @CHIP Ethernet MAC Address

```
unsigned char Get_Eth_Address( unsigned char *buffer,
                               unsigned int len );
```

## Parameters

*buffer*

Pointer to 6 byte buffer where the MAC Address should be stored

*len*

Length of the buffer (must be 6)

## Return Value

0

## See Also

Send_Packet

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## RTX_Wait_Sem_Priority

Wait with a specified priority on a semaphore.

```
int RTX_Wait_Sem_Priority ( int semID,
                            long time_ms,                                    unsigned
int wait_priority);
```

## Parameters

*semID*

The semaphore ID of a counting semaphore acquired by a call to **RTX_Create_Sem**.

*time_ms*

Maximum number of milliseconds to wait:

Zero ->Wait forever for semaphore
Positive value ->Maximum number of milliseconds to wait for resource
Negative value ->Polling mode, return immediately. If semaphore was not available the return value will be the -27 time-out indication.

*wait_priority*

The priority at which the caller wishes to wait (0 = highest). (Note that this priority has no connection to task priorities.) To wait in FIFO order, either have all callers use the same value here or use the alternative functions **RTX_Wait_Sem()** or **RTX_Reserve_Sem()**.

## Return Value

0 on success else **error** code.

## Comments

This function may be used with either counting or resource type semaphores.

This call waits for up to a specified time on a semaphore and returns with an error if the semaphore is still not available within this time limit. Otherwise the caller is granted the semaphore.

When more than one task is waiting for the semaphore, the task which called with the lowest `wait_priority` number (highest priority) will be granted the semaphore before of other waiting tasks. Tasks with matching `wait_priority` wait in FIFO order.

## See Also

RTX_Wait_Sem RTX_Get_Sem RTX_Signal_Sem
RTX_Reserve_Sem RTX_Release_Sem RTX_Free_Sem

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.22 | V1.22 | V1.22 | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---



# @CHIP-RTOS C Library - SPI API

Header File: `#include <SPIAPI.H>`

## Init SPI Init Functions

| | |
|---|---|
| spi_init | Init the SPI Interface (software emulation) |
| spi_init_hw | Init the SPI Interface (hardware interface) |

## Access SPI Access Functions

| | |
|---|---|
| spi_write | Write to the SPI Interface (software emulation/hardware interface) |
| spi_read | Read from the SPI Interface (software emulation/hardware interface) |
| spi_read_write | Read and write from/to the SPI Interface (software emulation/hardware interface) |
| spi_write_hw | Write to the SPI Interface (with return value for hardware interface) |
| spi_read_hw | Read from the SPI Interface (with return value for hardware interface) |
| spi_read_write_hw | Read and write from/to the SPI Interface (with return value for hardware interface) |
| spi_waiton_sem | Wait on SPI semaphore (hardware interface) |
| spi_waiton_sem_setmode | Wait on SPI semaphore, set SPI mode and divider (hardware interface) |
| spi_release_sem | Release the SPI semaphore (hardware interface) |

## Slave SPI Slave Functions

| | |
|---|---|
| spi_slave_init | Init the SPI Slave Interface (hardware interface) |
| spi_slave_write | Write to the SPI Slave Interface (hardware interface) |
| spi_slave_read | Read from the SPI Slave Interface (hardware interface) |
| spi_slave_read_write | Read and write from/to the SPI Slave Interface (hardware interface) |

List of C Libraries
@CHIP-RTOS Main Index




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS C Library - SPI API

---

## spi_init

Init the SPI Interface (software emulation)

```
void spi_init ( unsigned int mode, char clock,
                char dataIn, char dataOut );
```

## Parameters

*mode*

must be '0'

*clock*

Number of the pio which should be used as clock pin

*dataIn*

Number of the pio which should be used as data in pin

*dataOut*

Number of the pio which should be used as data out pin

## Return Value

-- none --

## See Also

**spi_write** **spi_read**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_init_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):
`#define SPI_DYN_LINK`

```
#include <clib.h>
```
This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_init_hw

Init the SPI Interface (hardware interface)

```
void spi_init_hw ( unsigned int mode, unsigned int div );
```

## Parameters

*mode*

Combine the following defines with a bitwise OR to select the operation mode.

SPI mode:
set one of these: `SPI_MODE0, SPI_MODE1, SPI_MODE2 or SPI_MODE3`
Open-Drain-Mode:
select `SPI_OPEN_DRAIN` for open-drain-mode,
else output mode is used
Autodrive:
select `SPI_AUTODRIVE` to autodrive SLVSEL chipselect pin,
works only together with SPI_ALTERNATE_IO
else do not drive SLVSEL pin
Invert chipselect:
select `SPI_INVERT_CS` to invert SLVSEL chipselect pin (active high),
else do not invert SLVSEL pin (active low)
Shift count:
select the shift count with these defines:
`SPI_SHIFT1, SPI_SHIFT2, SPI_SHIFT3, SPI_SHIFT4,`
`SPI_SHIFT5, SPI_SHIFT6, SPI_SHIFT7, SPI_SHIFT8`
Swap SDI and SDO:
select `SPI_ALTERNATE_IO` to swap SDI and SDO pins

*div*

SPI speed clock divider (valid values: 0-65535)
The SPI clock frequency is calculated with the formula:
SPI freq = CPU freq / (divider*2+2)

## Return Value

-- none --

## Comments

A Wired-OR mode is provided which allows multiple masters to collide on the bus without risk of damage. In this mode, an external pull-up resistor is required on the Master Out Slave In (MOSI) and Master In Slave Out (MISO) pins. Generally, pull-ups are recommended on all of the external SPI signals to ensure

they are held in a valid state, even when the SPI interface is disabled.

## See Also

spi_write_hw  spi_read_hw

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_init_hw_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS C Library - SPI API

---

## spi_write

Write to the SPI Interface

```
void spi_write ( const void far *bufferPtr, unsigned short nbytes );
```

## Parameters

*bufferPtr*

Pointer to buffer which contains write data

*nbytes*

number of bytes to write

## Return Value

-- none --

## See Also

spi_init  spi_read

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_write_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS C Library - SPI API

---

## spi_read

Read from the SPI Interface

```
void spi_read ( void far *bufferPtr, unsigned short nbytes );
```

## Parameters

*bufferPtr*

Pointer to buffer where read data should be stored

*nbytes*

number of bytes to read

## Return Value

-- none --

## See Also

spi_init  spi_write

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_read_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## spi_read_write

Read and write from the SPI Interface simultaneously

```
void spi_read_write ( void far *rbufferPtr, const void far *wbufferPtr,
                      unsigned short nbytes );
```

## Parameters

*rbufferPtr*

Pointer to buffer where read data should be stored

*wbufferPtr*

Pointer to buffer which contains write data

*nbytes*

number of bytes to read

## Return Value

-- none --

## See Also

spi_init  spi_write  spi_read

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_read_write_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_write_hw

Write to the SPI Interface (hardware interface)

```
unsigned int spi_write_hw ( const void far *bufferPtr,
                            unsigned short nbytes );
```

## Parameters

*bufferPtr*

Pointer to buffer which contains write data

*nbytes*

number of bytes to write

## Return Value

- 1, if illegal arguments were passed
- Else status bitmap:
  - BIT5 = 1: A collision has occurred between multiple masters
  - BIT6 = 1: An overrun was detected

## Comments

Use this function if the hardware interface is used.
Overrun detection only works if the SLVSEL pin is used as chipselect.

## See Also

spi_init_hw  spi_read_hw

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_write_hw_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
```

```
#include <clib.h>
```
This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_read_hw

Read from the SPI Interface (hardware interface)

```
unsigned int spi_read_hw ( void far *bufferPtr, unsigned short nbytes );
```

## Parameters

*bufferPtr*

Pointer to buffer where read data should be stored

*nbytes*

number of bytes to read

## Return Value

- 1, if illegal arguments were passed
- Else status bitmap:
  - BIT5 = 1: A collision has occurred between multiple masters
  - BIT6 = 1: An overrun was detected

## Comments

Use this function if the hardware interface is used.
Overrun detection only works if the SLVSEL pin is used as chipselect.

## See Also

spi_init_hw  spi_write_hw

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_read_hw_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_read_write_hw

Read and write from the SPI Interface simultaneously (hardware interface)

```
unsigned int spi_read_write_hw ( void far *rbufferPtr,
                                 const void far *wbufferPtr,
                                 unsigned short nbytes );
```

## Parameters

*rbufferPtr*

Pointer to buffer where read data should be stored

*wbufferPtr*

Pointer to buffer which contains write data

*nbytes*

number of bytes to read

## Return Value

- 1, if illegal arguments were passed
- Else status bitmap:
  - BIT5 = 1: A collision has occurred between multiple masters
  - BIT6 = 1: An overrun was detected

## Comments

Use this function if the hardware interface is used.
Overrun detection only works if the SLVSEL pin is used as chipselect.

## See Also

spi_init_hw  spi_write_hw  spi_read_hw

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function

which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_read_write_hw_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_waiton_sem

Wait on the SPI semaphore (hardware interface)

```
unsigned int spi_waiton_sem ( void );
```

## Return Value

0 on success else **error** code.

## Comments

If multiple tasks use the SPI hardware master interface to access different slaves through different slave chip select PIO pins, this call should be used to synchronise the SPI activity on the bus. The external MMC/SD driver examples uses this call since version V1.02.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_waiton_sem_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_waiton_sem_setmode

Wait on the SPI semaphore, set mode and divider (hardware interface)

```
unsigned int spi_waiton_sem_setmode ( unsigned int mode, unsigned int div );
```

## Parameters

*mode*

Combine the following defines with a bitwise OR to select the operation mode

SPI mode:
set on of these: `SPI_MODE0, SPI_MODE1, SPI_MODE2 or SPI_MODE3`
Open-Drain-Mode:
select `SPI_OPEN_DRAIN` for open-drain-mode,
else output mode is used
Autodrive:
select `SPI_AUTODRIVE` to autodrive SLVSEL chipselect pin,
works only together with SPI_ALTERNATE_IO
else do not drive SLVSEL pin
Invert chipselect:
select `SPI_INVERT_CS` to invert SLVSEL chipselect pin (active high),
else do not invert SLVSEL pin (active low)
Shift count:
select the shift count with these defines:
`SPI_SHIFT1, SPI_SHIFT2, SPI_SHIFT3, SPI_SHIFT4,`
`SPI_SHIFT5, SPI_SHIFT6, SPI_SHIFT7, SPI_SHIFT8`
Swap SDI and SDO:
select `SPI_ALTERNATE_IO` to swap SDI and SDO pins

*div*

SPI speed clock divider (valid values: 0-65535)
The SPI clock frequency is calculated with the formula:
SPI freq = CPU freq / (divider*2+2)

## Return Value

0 on success else **error** code.

## Comments

A Wired-OR mode is provided which allows multiple masters to collide on the bus without risk of damage. In this mode, an external pull-up resistor is required on the Master Out Slave In (MOSI) and Master In Slave Out (MISO) pins. Generally, pull-ups are recommended on all of the external SPI signals to ensure

they are held in a valid state, even when the SPI interface is disabled.

If multiple tasks use the SPI hardware master interface to access different slaves through different slave chip select PIO pins, this call should be used to synchronise the SPI activity on the bus. The external MMC/SD driver examples uses this call since version V1.02.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_waiton_sem_setmode_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_release_sem

Release the SPI semaphore (hardware interface)

```
unsigned int spi_release_sem ( void );
```

## Return Value

0 on success else **error** code.

## Comments

If multiple tasks use the SPI hardware master interface to access different slaves through different slave chip select PIO pins, this call should be used to synchronise the SPI activity on the bus. The external MMC/SD driver examples uses this call since version V1.02.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_release_sem_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_slave_init

Init the SPI Slave Interface (hardware interface)

```
void spi_slave_init ( unsigned int mode );
```

## Parameters

*mode*

Combine the following defines with a bitwise OR to select the operation mode

SPI mode:
set on of these: `SPI_MODE0, SPI_MODE1, SPI_MODE2 or SPI_MODE3`
Open-Drain-Mode:
select `SPI_OPEN_DRAIN` for open-drain-mode,
else output mode is used
Invert chipselect:
select `SPI_INVERT_CS` to invert SLVSEL chipselect pin (active high),
else do not invert SLVSEL pin (active low)
Swap SDI and SDO:
select `SPI_ALTERNATE_IO` to swap SDI and SDO pins

## Return Value

-- none --

## Comments

To use the SPI slave interface the user has to install an interrupt service routine for the SPI interrupt. An interrupt occurs when the SLVSEL pin transitions from active to inactive.

A Wired-OR mode is provided which allows multiple masters to collide on the bus without risk of damage. In this mode, an external pull-up resistor is required on the Master Out Slave In (MOSI) and Master In Slave Out (MISO) pins. Generally, pull-ups are recommended on all of the external SPI signals to ensure they are held in a valid state, even when the SPI interface is disabled.

## See Also

spi_slave_write  spi_slave_read

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_slave_init_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_slave_write

Write to the SPI Slave Interface (hardware interface)

```
void spi_slave_write ( unsigned char data );
```

## Parameters

*data*

Data to write

## Return Value

-- none --

## Comments

Use this function if the hardware slave interface is used.

## See Also

spi_slave_init  spi_slave_read

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_slave_write_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_slave_read

Read from the SPI Slave Interface (hardware interface)

```
unsigned char spi_slave_read ( void );
```

## Return Value

Read data

## Comments

Use this function if the hardware slave interface is used.

## See Also

spi_slave_init  spi_slave_write

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_slave_read_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|

V2.01

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## spi_slave_read_write

Read and write from the SPI Slave Interface simultaneously (hardware interface)

```
unsigned char spi_slave_read_write ( unsigned char data );
```

## Parameters

*data*

Data to write

## Return Value

Read data

## Comments

Use this function if the hardware slave interface is used.

## See Also

spi_slave_init  spi_slave_write  spi_slave_read

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `spi_slave_read_write_dyn`. You should however not use this function directly. Instead you should add the define `SPI_DYN_LINK` to your program before including the CLIB's header file(s):

```
#define SPI_DYN_LINK
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Get_IPConfig

Get IP, subnet mask and/or gateway of the default Ethernet interface.

```
void Get_IPConfig ( char far *IP,
                    char far *Mask,
                    char far *Gateway );
```

## Parameters

*IP*

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

*Mask*

Output parameter: Pointer to a 16 byte memory area where the subnet mask is to be stored as a null terminated string.

*Gateway*

Output parameter: Pointer to a 16 byte memory area where the gateway address is to be stored as a null terminated string.

## Return Value

-- none --

## Comments

Any of the three output parameters can be set to NULL pointers if the respective output is not desired.

## SC1x3/SC23 Comments

This function applies only to the IPv4 configuration of the internal integrated Ethernet device interface.

## See Also

Set_IPConfig  Get_IPv6_Iface_Entries

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Set_IPConfig

Set IP, subnet mask of the default Ethernet interface and/or gateway.

```
void Set_IPConfig ( const char far *IP,
                    const char far *Mask,
                    const char far *Gateway );
```

## Parameters

*IP*

Pointer to new IP address expressed as a null terminated ASCII string in network address dot notation.

*Mask*

Pointer to new net submask expressed as a null terminated ASCII string in dot notation.

*Gateway*

Pointer to new gateway address expressed as a null terminated ASCII string in dot notation.

## Return Value

-- none --

## Comments

Any of the three parameters can be set to NULL pointers if the respective value is not to be set.

## SC1x3/SC23 Comments

This function applies only to the IPv4 configuration of the internal integrated Ethernet device interface.

## See Also

PPP_Server_GetIP  PPP_Client_GetIP  Get_IPConfig
Reconfigure_ethernet

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Server_GetIP

Reads the IP Address of the PPP Server

```
void PPP_Server_GetIP (char far *IP );
```

## Parameters

*IP*

Output parameter: Pointer to a 16 byte memory area where the IPv4 address is to be stored as a null terminated string.

## Return Value

-- none --

## Comments

Copies the current IPv4 address of the PPP Server into the provided string. If no IP address of the PPP Server is available, the string will be empty.

## SC1x3/SC23 Comments

**This function is deprecated,** because it does not work if the PPP server uses the IPv6 protocol. The IP configuration of all TCP/IP devices can be read by using **Get_Iface_Entries** or **Get_IPv6_Iface_Entries**.

## See Also

**Get_IPConfig** **PPP_Client_GetIP**
**Get_Iface_Entries** **Get_IPv6_Iface_Entries**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.10** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Client_GetIP

Reads the IP Address of the PPP Client

```
void PPP_Client_GetIP ( const char far *IP );
```

## Parameters

*IP*

Output parameter: Pointer to a 16 byte memory area where the IPv4 address is to be stored as a null terminated string.

## Return Value

-- none --

## Comments

Copies the current IPv4 address of the PPP Client into the provided string. If no IP address of the PPP Client is available, the string will be empty.

## SC1x3/SC23 Comments

**This function is deprecated,** because it does not work if the PPP server uses the IPv6 protocol. The IP configuration of all TCP/IP devices can be read using either **Get_Iface_Entries** or **Get_IPv6_Iface_Entries**.

## See Also

**Get_IPConfig** **PPP_Server_GetIP**
**Get_Iface_Entries** **Get_IPv6_Iface_Entries**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.10** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## InetToAscii

Convert an IP address to an ASCII dotted decimal IP string.

```
int InetToAscii ( const unsigned long far *IPAddress,
                  char far *IPAddressStringPtr );
```

## Parameters

*IPAddress*

Pointer to the 32 bit IP address in network byte order ("big endian").

*IPAddressStringPtr*

Output parameter: Pointer to a **16 character buffer** (minimum) where a null terminated ASCII string will be written by this API function containing the IP address expressed in dotted decimal notation.

## Return Value

Always 0

## Comments

This buffer at `IPAddressStringPtr` must have space for 16 Bytes!.

## SC1x3/SC23 Comments

This function applies only to IPv4 addresses. The **inet_ntop** API can be used for either IPv4 or IPv6 address conversions.

## See Also

**inet_addr** **inet_ntop**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## inet_addr

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

```
int inet_addr ( const char far *IPAddressStringPtr,
                unsigned long far *IPAddress );
```

## Parameters

*IPAddressStringPtr*

Pointer to null terminated ASCII string of network address in dot notation.

*IPAddress*

Output parameter: Pointer to a 32 bit location where binary equivalent to the provided ASCII IP address will be output.

## Return Value

0: success
non-zero: Invalid IP address.

## SC1x3/SC23 Comments

This function applies only to IPv4 addresses. The **inet_pton** API can be used for either IPv4 or IPv6 address conversions.

## See Also

**tcp_connect** **InetToAscii** **inet_pton**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## inet_pton

Convert an IP address from standard text representation to binary.

```
int inet_pton( int af,
               const char far * source_ptr,
               void far * dest_ptr);
```

## Parameters

*af*

Address family, AF_INET (for IPv4) or AF_INET6 (for IPv6)

*src*

Input Parameter: Pointer to the IP address text (null terminated) to be converted

*dst*

Output Parameter: Storage provided by the user which will receive the converted binary IP address. This buffer must provide 4 bytes for AF_INET and 16 Bytes for AF_INET6.

## Return Value

1: Success, the `dst` buffer contains the converted binary IP in network byte order.
0: Input string at `src` was not a valid IP address of indicated type (IPv4 or IPv6).
-1: The `af` parameter was set to an invalid address family.

## See Also

inet_ntop inet_addr

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## inet_ntop

Convert an IP address from binary to standard text.

```
int inet_ntop( int af,
               const void far * source_ptr,
               char far * dest_ptr,int max);
```

## Parameters

*af*

Address family, AF_INET (for IPv4) or AF_INET6 (for IPv6)

*src*

Input Parameter: Pointer to the IP address in binary to be converted.

*dst*

Output Parameter: Storage provided by the caller which will receive the converted IP address text. This text will be a null terminated string.

*max*

Size in bytes, of the provided buffer at `dst`. This string buffer should provide 16 bytes for AF_INET and 46 Bytes for AF_INET6.

## Return Value

1: Success, the `dst` buffer contains the converted IP address string.
0: Input at `src` was not a valid IP address of indicated type (IPv4 or IPv6).
-1: The `af` parameter was set to an invalid address family.

## See Also

inet_pton  InetToAscii

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## opensocket

Open a socket.

```
int opensocket ( unsigned char type, int *error );
```

## Parameters

*type*

SOCK_STREAM = 1 for TCP socket
SOCK_DGRAM = 2 for UDP datagram socket.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1: Failure (see `error` output parameter)
else socket descriptor

## Comments

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

## SC1x3/SC23 Comments

This function applies only to IPv4 sockets. The **opensocket_ext** API may be used to open either IPv4 or IPv6 sockets.

## See Also

**closesocket** **bind** **connect**
**opensocket_ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## opensocket_ext

Open an IPv4 or IPv6 socket.

```
int opensocket_ext(int family, unsigned char type, int *error);
```

## Parameters

*family*

address family AF_INET (2) for IPv4 or AF_INET6 (28) for IPv6

*type*

SOCK_STREAM = 1 for TCP socket
SOCK_DGRAM = 2 for UDP datagram socket.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1: Failure (see `error` output parameter)
else socket descriptor

## Comments

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

## See Also

**closesocket** **bind** **connect**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## closesocket

Close a socket.

```
int closesocket ( int sd, int *error );
```

## Parameters

*sd*

Socket descriptor.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1: Failure (see `error` output parameter)
0: success

## Comments

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

## See Also

**opensocket** **opensocket_ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## bind

Bind an unnamed socket to an address and port number.

```
int bind ( int sd,
           const struct sockaddr far *addressPtr,
           int *error );
```

## Parameters

*sd*

Socket descriptor.

*addressPtr*

Pointer to a **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure preset by caller.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

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

The caller must fill in the **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure at `addressPtr` prior to making this API call. An example (IPv4) of how this can be done follows. This function's prototype uses a pointer to the generic type `sockaddr` for its `addressPtr` parameter, for compatibility between IPv4 and IPv6 protocols. The pointer to the `sockaddr_in` (IPv4) or `sockaddr_in6` (IPv6) data structure which is actually used should be cast to this type as shown below to avoid compiler warnings, "Suspicious pointer conversion".

```
// Example
int ClientPort = 4000 ;
struct sockaddr_in addr ;
int error_code ;

addr.sin_family = PF_INET ;
addr.sin_addr.s_addr =  0 ;
addr.sin_port = htons(ClientPort);   // convert byte order
error_code = bind (sd, (const struct sockaddr *)&addr, &error_code) ;
```

## See Also

**opensocket** **opensocket_ext** **connect**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## connect

Connect to another socket.

```
int connect ( int sd,
              const struct sockaddr far *addressPtr,
              int *error );
```

## Parameters

*sd*

Socket descriptor.

*addressPtr*

Pointer to a **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure containing host's IP address and port number.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

The caller must fill in the **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure at `addressPtr` prior to calling here. An example of how this normally would be done follows. The PF_INET setting for the `sin_family` is required for IPv4. For IPv6 set `sin6_family` to PF_INET6.

This function's prototype uses a pointer to the generic type `sockaddr` for its `addressPtr` parameter, for compatibility between IPv4 and IPv6 protocols. The pointer to the `sockaddr_in` or `sockaddr_in6` data structure which is actually used should be cast to this type as shown below to avoid compiler warnings, "Suspicious pointer conversion".

```
// Example
char szHostIPStr[] = "172.30.1.68" ;
int HostPort = 4000 ;
struct sockaddr_in addr ;
int error_code ;

addr.sin_family = PF_INET ;
addr.sin_addr.s_addr =  0 ;
addr.sin_port = htons (HostPort) ;   // convert byte order

// convert server IP address string to binary
inet_addr (szHostIPStr, &addr.sin_addr.s_addr);
// establish a connection to the server
error_code = connect (sd, (const struct sockaddr far *)&addr) ;
```

## See Also

**opensocket** **tcp_connect** **htons**
**opensocket_ext** **inet_addr** **bind**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## GetWaitingBytes

Report the number of bytes available on a socket, waiting to be read.

```
int GetWaitingBytes ( int sd, int *error );
```

## Parameters

*sd*

Socket descriptor.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1 on failure, see error reported
else count of received bytes waiting on socket.

## See Also

**recv** **recvfrom** **opensocket**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## Set_IP_Type_Of_Service

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

```
int Set_IP_Type_Of_Service ( int sd,
                             unsigned char tos,
                             int *error );
```

## Parameters

*sd*

Socket descriptor.

*tos*

Type-Of-Service for IP datagrams

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## 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.

## See Also

**opensocket**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## setsockopt

Set options on a socket.

```
int setsockopt ( int sd,
                 const SetSocketOption far *sockoptptr,
                 int *error);
```

## Parameters

*sd*

Socket descriptor.

*sockoptptr*

Pointer to SetSocketOption **type** that specifies the socket options (see tcpipapi.h).

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## 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. The socket options of an incoming connection (using **accept**) will be the same as the socket options of the listening socket.

## See Also

**opensocket** **accept** **getsockopt**
**opensocket_ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## getsockopt

Get options on a socket.

```
int getsockopt ( int sd,
                 const GetSocketOption far *sockoptptr,
                 int *error );
```

## Parameters

*sd*

Socket descriptor.

*sockoptptr*

Pointer to GetSocketOption **type** that specifies the socket options (see tcpipapi.h) and contains pointers to where the specified socket option will be reported to.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

On success, this function reports the requested socket option value to the buffer pointed to by the `optionValue` member of GetSocketOption type at `sockoptptr` and the length of this reported value is written to the location pointed to by the `optionLen` member.

The caller must setup the GetSocketOption **type** structure prior to calling. In particular, the `optionValue` member must point to a sufficiently large buffer to receive the reported option value and the size of this buffer must be stated in the location referenced by the `optionLen` member. Note that this length specifier is both an input and an output parameter for this function. If on entry, the location referenced by `optionLen` contains a value less than the buffer space required to report the requested socket option value, then the function will fail and return error code 242.

The socket options of an incoming connection (using **accept**) will be the same as the socket options of the listening socket.

## See Also

**opensocket** **setsockopt**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Set_Blocking_Mode

Set a socket in blocking or non-blocking mode.

```
int Set_Blocking_Mode ( int sd,
                        unsigned char blocking_mode,
                        int *error );
```

## Parameters

*sd*

Socket descriptor.

*blocking_mode*

0: switch blocking off, 1:switch blocking on.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

By default all sockets are in blocking mode. If a socket is set to non blocking mode, socket calls such as **connect** and **accept** do not wait until full completion. Instead they return immediately.

Example usage of non-blocking mode:

The **connect**call returns at a non blocking socket with error code 236 or 237, when the connection was not completed. The user can repeat this API call in a periodic loop until a successful connection is established. In this case the connect function returns error code 0 or 256 (socket is already connected).

## See Also

**connect** **accept** **recv**
**recvfrom** **send** **sendto**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## get_socketerror

Get last socket error.

```
int get_socketerror ( int sd );
```

## Parameters

*sd*

Socket descriptor.

## Return Value

**error** code

## Comments

Returns the last error which occured on specified socket.

## See Also

**opensocket** **opensocket_ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## tcp_connect

Open a client TCP/IP socket connection to a specified host.

```
int tcp_connect ( const char far *DestIPStr,
                  unsigned int ClientPort,
                  unsigned int HostPort,
                  int *error );
```

## Parameters

*DestIPStr*

Pointer to server's IPv4 address, expressed as a null terminated ASCII string in network dot notation.

*ClientPort*

Optional local port number to assign to socket. Set this to zero if a bind operation is not desired (random port).

*HostPort*

Remote port number to which connection is desired.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1 = Failure (see `error` output parameter)
else new socket descriptor

## Comments

This function calls the required set of TCP/IP functions to establish a IPv4 client connection to a host.

Pseudo code:

```
struct sockaddr_in addr;
int sd = opensocket(SOCK_STREAM, error) ;
addr.sin_family = PF_INET ;
addr.sin_addr.s_addr = 0 ;
IF ClientPort != 0 THEN
    addr.sin_port = htons(ClientPort) ;
    bind(sd, &addr, error) ;
```

```
ENDIF
addr.sin_port = htons(HostPort) ;
error = inet_addr(DestIPStr, &addr.sin_addr.s_addr) ;
error = connect (sd, (const struct sockaddr *)&addr) ;
return sd ;
```

The error handling is not shown in the above simplified pseudo code. In case of errors, the socket is closed by this API function and a return is made from the point of the error with return value -1. On success, the new socket descriptor is returned. The caller is then responsible for calling **closesocket** when finished with the connection.

## SC1x3/SC23 Comments

The actual library function reached here is named `tcp_connect_ipstackV2`. The name change occurs due to a macro in the library header file.

## See Also

**opensocket** **htons** **bind**
**inet_addr** **connect** **closesocket**

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## accept

Accept the next incoming connection.

```
int accept ( int sd,
             struct sockaddr far *addressPtr,
             int *error );
```

## Parameters

*sd*

Socket descriptor (must be switched into listen mode using **listen** API).

*addressPtr*

Optional output parameter: Pointer to a **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure. On success, this function fills in this sockaddr_in or sockaddr_in6 structure with the IP address and port number of the accepted connection. This pointer can be set to NULL if this address and port number report is not desired.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1: Failure, error code output to `error`
otherwise: New socket descriptor for the connection

## Comments

This call is used by a TCP server.

This API function extracts the first connection on the queue of pending connections. It then creates a new socket for this connection. This new socket will have the same socket options as the listening socket, `sd`.

This function's prototype uses a pointer to the generic type `sockaddr` for its `addressPtr` parameter, for compatibility between IPv4 and IPv6 protocols. The pointer to the `sockaddr_in` (IPv4) or `sockaddr_in6` (IPv6, *SC1x3/SC23 only!* ) data structure which is actually used should be cast to `sockaddr` pointer type here to avoid compiler warnings, "Suspicious pointer conversion".

## See Also

listen opensocket opensocket_ext

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## listen

Listen for incoming connections.

```
int listen ( int sd, int backlog, int *error );
```

## Parameters

*sd*

Socket descriptor.

*backlog*

The maximum number (limited to 5) of allowed outstanding connections

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

This API function places the socket in passive mode and sets the number of incoming TCP connections that the system will queue (`backlog` ).

This call is used by a TCP server.

## See Also

**accept** **opensocket** **opensocket_ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS C Library - TCP/IP API

## htons

Converts a 16 bit word from host to TCP/IP network byte order.

```
unsigned short htons ( unsigned short hostshort );
```

## Parameters

*hostshort*

Port number in little endian (PC) byte order

## Return Value

input value returned in network byte order (big endian)

## Comments

This function converts the provided 16 bit value from Intel byte ordering (little endian) to network byte order (big endian). This is commonly needed for TCP/IP port number specifications.

No entry into the @CHIP-RTOS API is made here. This action is taken immediately within this library call.

## See Also

**connect** **bind**

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## send

Transmit a message.

```
int send ( int sd, const char far *bufptr, int bufLen,
           int flags, int *error );
```

## Parameters

*sd*

Socket descriptor.

*bufptr*

Pointer to output buffer containing data to be transmitted.

*bufLen*

Maximum characters to output from `bufptr` .

*flags*

Wait option:

- **MSG_BLOCKING** - Wait for data send to complete.
- **MSG_DONTWAIT** - Don't wait for data send to complete.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1: Failure, see error value reported
otherwise: Count of bytes transfered into the socket send queue

## Comments

This API function applies only to TCP sockets. It transmits a message to another transport end-point. It may be used only if the socket is in a connected state.

The MSG_BLOCKING mode will not work if the socket has been put into non-blocking mode using the **Set_Blocking_Mode** API.

## See Also

**recv** **sendto** **opensocket**
**Set_Blocking_Mode**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## recv

Receive a message from another socket.

```
int recv ( int sd, char far *bufptr, int bufLen,
           int flags, unsigned long timeout,
           int *error );
```

## Parameters

*sd*

Socket descriptor.

*bufptr*

Output parameter: Pointer to input buffer where received characters will be stored.

*bufLen*

Maximum characters to store in buffer at `bufptr` .

*flags*

Wait option:

- **MSG_BLOCKING** - Sleep until data comes in.
- **MSG_TIMEOUT** - The caller wakes up after timeout or if `bufLen` data is received.
- **MSG_DONTWAIT** - Return immediately after collecting what ever data is ready.

*timeout*

Maximum milliseconds to wait if `flags` is set to MSG_TIMEOUT

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1: Failure, see error value reported
otherwise: Count of received bytes stored in buffer at `bufptr`

## Comments

This API function applies only to TCP sockets. It receives a message from another socket. This API

function may be used only if the socket is in a connected state.

If `flags` parameter is set to MSG_DONTWAIT, this API call returns immediately. If no data is available -1 is returned and output parameter `error` is set to 235.

If `flags` parameter is set to MSG_BLOCKING, the `recv` call waits for a message to arrive. By default the blocking mode is set for all sockets at the **opensocket** call. If however a socket was subsequently set to non blocking mode with the **Set_Blocking_Mode** API, the MSG_BLOCKING flag will then not work here.

## See Also

**send** **recvfrom** **opensocket**
**Set_Blocking_Mode** **select** **select_ms**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## ResetConnection

Abort a connection on a TCP socket.

```
int ResetConnection ( int sd, int *error );
```

## Parameters

*sd*

Socket descriptor.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

This API function only works on TCP sockets.

A reset socket cannot be used for a new connection. You must close the socket and open a new one.

This API function does not close the socket, This must be done using the **closesocket** API.

## See Also

**closesocket** **opensocket** **opensocket_ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## setlinger

Set linger time on close (TCP only).

```
int setlinger ( int sd, int seconds, int *error );
```

## Parameters

*sd*

Socket descriptor.

*seconds*

Linger time in seconds, default: 20 sec, 0 means linger turned off.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

This API function works only with TCP sockets.

## See Also

**opensocket** **opensocket_ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## setreuse

Set reuse option on a listening socket.

```
int setreuse ( int sd, int *error );
```

## Parameters

*sd*

Socket descriptor.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

This API function works only with TCP sockets. In earlier SC12 @CHIP-RTOS versions, this API was necessary when a listening socket was closed and was to be re-opened and bound to the same port as it was bound to before.

Since SC12 @CHIP-RTOS 071, every TCP socket is set by default as reusable so use of this API function is no longer necessary.

## See Also

**opensocket** **bind**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.00** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Get_TCP_Socket_State

Find TCP socket and return its state.

```
unsigned char Get_TCP_Socket_State ( unsigned int localPort,
                                     unsigned long far *remoteIP,
                                     unsigned int *remotePort );
```

## Parameters

*localPort*

Local port (e.g. htons(23) for telnet).

*remoteIP*

Output parameter: Pointer to storage for 32 bit remote IPv4 address.

*remotePort*

Output parameter: Pointer to storage for 16 bit remote peer port number.

## Return Value

Socket state (see list below)

## Comments

This API searches for a TCP socket by a given local port number, returning the socket state, IP address and port number of the remote peer (if any).

If the return value is not INVALID (=20) and is greater or equal to SYN_SENT (=2) then a TCP connection is established. In this case the storage at `remoteIP` holds the 32 bit IP address of the connected remote peer and `remotePort` contains the remote peer port number.

Possible TCP socket states:

0: CLOSED
1: LISTEN
2: SYN_SENT
3: SYN_RECEIVED
4: ESTABLISHED
5: CLOSE_WAIT
6: FIN_WAIT_1
7: CLOSING

8: LAST_ACK
9: FIN_WAIT_2
10: TIME_WAIT
20: INVALID

## SC1x3/SC23 Comments

Deprecated, because only IPv4 addresses can be stored at the provided 32-bit buffer at `remoteIP` . The alternative **Get_TCP_Socket_State_IP_All** API is instead recommended, which covers both IPv4 and IPv6 protocols.

## See Also

**opensocket** **htons** **Get_TCP_Socket_State_IP_All**
**opensocket_ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.04 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Get_TCP_Socket_State_IP_All

Find TCP socket and return its state.

```
unsigned char Get_TCP_Socket_State_IP_All (
                              unsigned int localPort,
                              struct in6_addr far *remoteIP,
                              unsigned int *remotePort );
```

## Parameters

*localPort*

Local port (e.g. htons(23) for telnet).

*remoteIP*

Output parameter: Pointer to a **in6_addr** structure which will receive the remote IP address.

*remotePort*

Output parameter: Pointer to storage for 16 bit remote peer port number.

## Return Value

Socket state (see list below)

## Comments

Search for a TCP socket by a given local port number, returning the socket state, IP address and port number of the remote peer (if any).

If the return value is not INVALID (=20) and is greater or equal to SYN_SENT (=2) then a TCP connection is established. In this case the storage at `remoteIP` holds the IP address of the connected remote peer and `remotePort` contains the remote peer port number.

Possible TCP socket states:

```
CLOSED          0
LISTEN          1
SYN_SENT        2
SYN_RECEIVED    3
ESTABLISHED     4
CLOSE_WAIT      5
FIN_WAIT_1      6
CLOSING         7
LAST_ACK        8
FIN_WAIT_2      9
TIME_WAIT      10
INVALID        20
```

## See Also

**opensocket** **Get_TCP_Socket_State** **htons**
**opensocket_ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## int FindAllOpenSockets

Find/return information about all open sockets listed in the internal socket table.

```
int FindAllOpenSockets( SocketInfo far * SockInfoPtr,
                        int max);
```

## Parameters

*SockInfoPtr*

Output Parameter: Pointer to an array of `max` **SocketInfo** structures in the user's memory space into which this API will report the socket information.

*max*

Size of the array (maximum number of table entries reported).

## Return Value

Returns the number of open sockets. The user provided array holds the socket information for up to `max` sockets (the lesser of two counts: Return value, `max`).

## SC1x3/SC23 Comments

Deprecated, because only IPv4 addresses can be stored at the provided 32-bit IP address buffer in the `SocketInfo` structure.

Use **FindAllOpenSockets_IP_All** instead.

## See Also

**FindAllOpenSockets_IP_All**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.10** | **V1.00** | **V1.00** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## int FindAllOpenSockets_IP_All

Find/return information about all open sockets listed in the internal socket table. (IPv4 and IPv6)

```
int FindAllOpenSockets_IP_All(
                        SocketInfoIPv4v6 far * SockInfoPtr,
                        int max);
```

## Parameters

*SockInfoPtr*

Output Parameter: Pointer to an array of `max` **SocketInfoIPv4v6** structures in the user's memory space into which this API will report the socket information.

*max*

Size of the array (maximum number of table entries reported).

## Return Value

Returns the number of all open sockets. The user provided array holds the socket information for up to `max` sockets (the lesser of two counts: Return value, `max`).

## See Also

**FindAllOpenSockets**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## Shutdown

Shutdown a socket in read,write or both directions.

```
int shutdown ( int sd, unsigned char dir, int *error );
```

## Parameters

*sd*

Socket descriptor.

*dir*

Direction: 0:read, 1:write, 2: both

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

This API function only works on TCP sockets.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.01 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## recvfrom

Receive message from another socket.

```
int recvfrom ( int sd, char far *bufptr, int bufLen,
               int flags, unsigned long timeout,
               struct sockaddr far *fromPtr,
               int *error );
```

## Parameters

*sd*

Socket descriptor.

*bufptr*

Pointer to input buffer where received characters will be output.

*bufLen*

Maximum characters to store in buffer at `bufptr` .

*flags*

Wait option:

- **MSG_BLOCKING** - Sleep until data comes in
- **MSG_TIMEOUT** - The caller wakes up after timeout or if `bufLen` data bytes are received.
- **MSG_DONTWAIT** - Return immediately after collecting what ever data is ready

*timeout*

Maximum milliseconds to wait if `flags` is set to MSG_TIMEOUT:

*fromPtr*

Output parameter: Pointer to a **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure, which will be set to indicate from where the received data originated. This pointer can be set to NULL if this information is not desired.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1: Failure, see error value reported
otherwise: Count of bytes received and placed in `bufptr` (0 if time-out)

## Comments

This API function applies only to UDP sockets.

The MSG_BLOCKING mode will not work if the socket has been put into non-blocking mode using the **Set_Blocking_Mode** API.

This function's prototype takes a pointer to the generic type `sockaddr` for its `fromPtr` parameter, for compatibility between IPv4 and IPv6 protocols. The pointer to the `sockaddr_in` (IPv4) or `sockaddr_in6` (IPv6, *SC1x3/SC23 only!* ) data structure which is actually used should be cast to a `sockaddr` type pointer to avoid compiler warnings, "Suspicious pointer conversion".

## SC1x3/SC23 Comments

The actual library function reached here is named `recvfrom_ipstackV2` . The name change occurs due to a macro in the library header file.

## See Also

**sendto** **opensocket** **Set_Blocking_Mode**
**opensocket_ext** **select** **select_ms**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## recvfromto

Same as recvfrom, but returns the destination address

```
int recvfromto ( int sd, char far *bufptr, int bufLen,
                 int flags, unsigned long timeout,
                 struct sockaddr far *fromPtr,
                 struct sockaddr far *toPtr,
                 int *error );
```

## Parameters

*sd*

Socket descriptor.

*bufptr*

Pointer to input buffer where received characters will be output.

*bufLen*

Maximum characters to store in buffer at `bufptr` .

*flags*

Wait option:

- **MSG_BLOCKING** - Sleep until data comes in
- **MSG_TIMEOUT** - The caller wakes up after timeout or if `bufLen` data bytes are received.
- **MSG_DONTWAIT** - Return immediately after collecting what ever data is ready

*timeout*

Maximum milliseconds to wait if `flags` is set to MSG_TIMEOUT:

*fromPtr*

Output parameter: Pointer to a **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure, which will be set to indicate from where the received data originated. This pointer can be set to NULL if this information is not desired.

*toPtr*

Output parameter: Pointer to a user provided storage from type **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure, which will contain (after sucessful return) the destination address.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1: Failure, see error value reported
otherwise: Count of bytes received and placed in `bufptr` (0 if time-out). Storage pointed by toPtr holds the destination address.
occurs due to a macro in the library header file.

## See Also

**recvfrom** **select** **select_ms**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## sendto

Transmit a datagram.

```
int sendto ( int sd, const char far *bufptr, int bufLen,
             int flags, const struct sockaddr far *toPtr,
             int *error );
```

## Parameters

*sd*

Socket descriptor.

*bufptr*

Pointer to output buffer containing characters to be transmitted.

*bufLen*

Maximum characters to output from `bufptr` .

*flags*

Wait option:
- **MSG_BLOCKING** - Wait for data send to complete.
- **MSG_DONTWAIT** - Don't wait for data send to complete.

*toPtr*

Data destination, pointer to a **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-1: Failure, see error value reported
otherwise: Count of bytes transfered into the socket send queue.

## Comments

This API function applies only to UDP sockets. This function will output up to `bufLen` bytes from the

buffer at `bufptr` to the IP address specified by the **sockaddr_in** (IPv4) structure or the **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) structure referenced by the `toPtr` parameter.

This function's prototype takes a pointer to the generic type `sockaddr` for its `toPtr` parameter, for compatibility between IPv4 and IPv6 protocols. The pointer to the `sockaddr_in` (IPv4) or `sockaddr_in6` (IPv6, *SC1x3/SC23 only!* ) data structure which is actually used should be cast to a `sockaddr` type pointer to avoid compiler warnings, "Suspicious pointer conversion".

The MSG_BLOCKING mode will not work if the socket has been put into non-blocking mode using the **Set_Blocking_Mode** API.

## SC1x3/SC23 Comments

The actual library function reached here is named `sendto_ipstackV2` . The name change occurs due to a macro in the library header file.

## See Also

**recvfrom** **opensocket** **Set_Blocking_Mode**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## sendto_iface

Transmit a datagram via specified interface.

```
int sendto_iface ( int sd, const char far *bufptr, int bufLen,
            int flags, const struct sockaddr far *toPtr,
            DevUserIfaceHandle DevHandlePtr,
            int *error );
```

## Parameters

*sd*

Socket descriptor.

*bufptr*

Pointer to output buffer containing characters to be transmitted.

*bufLen*

Maximum characters to output from `bufptr` .

*flags*

Wait option:
- **MSG_BLOCKING** - Wait for data send to complete.
- **MSG_DONTWAIT** - Don't wait for data send to complete.

*toPtr*

Data destination, pointer to a **sockaddr_in** (IPv4) or **sockaddr_in6** (IPv6, *SC1x3/SC23 only!* ) data structure.

*DevHandlePtr*

Driver handle pointer **Device handle**, which caller must fill in prior to call. TCPIP send the packet via this specified Device handle.

*error*

Output parameter:  Failure **code**, 0 on success.

## Return Value

-1: Failure, see error value reported

otherwise: Count of bytes transfered into the socket send queue.

## Comments

This API function applies only to UDP sockets and works similiar than the **sendto** function. It is required, when the user wants to send a datagram with a multicast destination address via a specified device interface. It is not necessary to use this function, if the packet should by send via the default ethernet interface of the IPC@CHIP®. The existing API function **sendto** function does this by default, if a the multicast destination address is specified.

## SC1x3/SC23 Comments

The actual library function reached here is named `sendto_ipstackV2`. The name change occurs due to a macro in the library header file.

## See Also

**sendto** **Get driver handle pointer**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.21 | V1.21 | V1.21 | V1.10 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PKI_CertificateAdd

Add a certificate to the public key infrastructure. The CA certificate must be added before the certificate is added. In addition, the private/public key pair for a local certificate must be added before adding the local certificate.

```
int PKI_CertificateAdd(PKI_CERT_ADD_EntryPtr addEntryPtr, int *error);
```

## Parameters

*addEntryPtr*

Certificate Add structure

*error*

Output parameter: Failure code, 0 on success.

## Return Value

0: success
-1: failed

## See Also

PKI_CertificateDel  PKI_OwnKeyPairAdd

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|

| V2.02 |
|---|

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PKI_CertificateDel

Delete a certificate from the public key infrastructure.

```
int PKI_CertificateDel(void far *idPtr, int *error);
```

## Parameters

*idPtr*

pointer to certificate identification string

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: failed

## See Also

**PKI_CertificateAdd** **PKI_OwnKeyPairAdd**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PKI_OwnKeyPairAdd

Add a public/private key pair for a local certificate to the public key infrastructure.

```
int PKI_OwnKeyPairAdd(PKI_OWNKEYPAIR_ADD_EntryPtr keyEntryPtr,
                      int *error);
```

## Parameters

*keyEntryPtr*

Own key pair structure

*error*

Output parameter: Failure code, 0 on success.

## Return Value

0: success
-1: failed

## See Also

PKI_CertificateAdd PKI_CertificateDel

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSL_SessionNew

Open a new SSL session. Could be used as either SSL server session or SSL client session. If it is used as a server session the field certIdentity in the structure should be non-null, otherwise the SSL server can't run on this session.

```
int SSL_SessionNew(SSL_SESSION_NEW_EntryPtr newSessionPtr,
                   int *error);
```

## Parameters

*newSessionPtr*

New SSL structure

*error*

Output parameter: Failure code, 0 on success.

## Return Value

>0: Session ID
-1: failed

## Comments

Each session maintains its own connect information for the convenience of resumption. You can resume a previous connection only if the current connect attempt and the previous one are in the same SSL session, and your previous connection cache hasn't been overwritten yet.

For SSL server, all connections spawned from a single listening socket belong to one session, you should assign the session ID to the listening socket. All accepted sockets will inherit that session ID.

For SSL client, user determines which session one connection belongs to. For example, users may decide that any connection to the same SSL server uses a single session ID, so that when user is trying to connect to the same server again, he may resume the previous connection.

## See Also

SSL_SessionProp SSL_SessionOpt SSL_SessionClose

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSL_SessionClose

Close a SSL session.

```
int SSL_SessionClose(int sessionID, int *error);
```

## Parameters

*sessionId*

Session ID

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: Success
-1: failed

## Comments

All cached connection information in this session will be freed too. User should close the SSL connection or the socket, first.

## See Also

**SSL_SessionNew**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSL_SessionProp

Use this function to set proposals for a SSL session. For a client session, the proposals are used to construct the Client Hello message. For a server session, the proposals are used to match Client Hello message in order to determine the final cipher suite to use.

```
int SSL_SessionProp(int sessionID, int far *proposals, int count,
                    int *error);
```

## Parameters

*sessionID*

Session ID for SSL session

*proposals*

Array of integer with Session proposals

*count*

Number of entries in the proposal array

*error*

Output parameter:  Failure **code**, 0 on success.

## Return Value

0: Suceess
-1: failed

## Comments

To use this function on a server session, a previous call to function **SSL_SessionOpt** is needed. If the user doesn't call this function to set any session proposals, the default value is going to be used.

**Possible proposals array entries:**

```
TLS_RSA_NULL_MD5           = 0x0001
TLS_RSA_NULL_SHA           = 0x0002
TLS_RSA_EXPORT_RC4_40_MD5 = 0x0003
TLS_RSA_RC4_128_MD5        = 0x0004
TLS_RSA_RC4_128_SHA        = 0x0005
TLS_RSA_3DES_EDE_CBC_SHA   = 0x000A
TLS_RSA_AES_128_CBC_SHA    = 0x002F
TLS_RSA_AES_256_CBC_SHA    = 0x0035
TLS_RSA_EPT1K_RC4_56_SHA   = 0x0064
```

## See Also

SSL_SessionNew  SSL_SessionOpt

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.07 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## SSL_SessionOpt

Call this function to set SSL session options.

```
int SSL_SessionOpt(int sessionID, int option, int value,
                   int *error);
```

## Parameters

*sessionID*

Session ID for SSL session

*option*

option
1: Client Authentication
2: Server Proposals

*value*

option value
0: disable option
1: enable option

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: Suceess
-1: failed

## Comments

The client authentication option is used to enable client authentication on a SSL server session. It takes effect only if the session is bound to an SSL server socket, in which case, upon receiving the ClientHello message, the SSL server is going to send back a CertificateRequest message along with its own Certificate, ServerHello and ServerHelloDone messages.

The server proposals option is used to allow the SSL server to have more control on which cipher suite to use. If this option is not used on an SSL server session, upon receiving a ClientHello message, the SSL server just chooses the first matched cipher suite from its own session proposal list, according to server's preference. If a cipher suite proposed by the client is not the SSL server session's proposal list, even if the SSL server supports that cipher suite, it won't be chosen. For example, if server proposals option is

enabled and server prefers cipher suites 1,2,3,4 and in this order. However, client only offers 4,3,2 and in this order, then cipher suite 2 will be chosen.

## See Also

SSL_SessionNew  SSL_SessionProp  SSL_AddClientCA

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSL_AddClientCA

Call this function to add client CAs to ServerHello message.

```
int SSL_AddClientCA(int sessionID, void far *caIdPtr,
                    int *error);
```

## Parameters

*sessionID*

Session ID for SSL session

*caIdPtr*

Pointer to client CA certificate identity name

*error*

Output parameter:  Failure **code**, 0 on success.

## Return Value

0: Suceess
-1: failed

## Comments

If client authentication option is used to enable client authentication on a SSL server session, this call can be used to add CAs to the ServerHello message (called CertificateRequest).
This function must be called against an SSL server session. The SSL server will list CA's Distinguished Name to the client in his ServerHello message . This function can be called multiple times to add multiple potential CAs for client certificates.
Before you can use this call, the client CAs certificate must be added to the PKI.

## See Also

**PKI_CertificateAdd** **SSL_SessionNew** **SSL_SessionOpt**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSL_ClientStart

Call this function to start the SSL client and SSL handshaking. Before this call, the user should set proper socket options on socket descriptor.

```
int SSL_ClientStart(int sd, char far * srvCommonName, int *error);
```

## Parameters

*sd*

Socket descriptor

*srvCommonName*

Server common name.
The common name is used to verify the certificate of the peer. If user passes in a NULL string, the certificate identity is not checked. This means, any valid certificate, even if it belongs to a bad party, will be accepted.

*error*

Output parameter:  Failure **code**, 0 on success.

## Return Value

0: Success
-1: failed

## Comments

The certificate identity (common name field) may contain the wildcard character which is considered to match any single domain name component or component fragment. E.g., "*.a.com" in certificate Common Name matches the server common name "foo.a.com" but not "bar.foo.a.com". And certificate common name "f*.com" matches server common name "foo.com" but not "bar.com".

## See Also

**SSL_SessionNew** **setsockopt**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSL_SendFlush

Queued SSL user data will be flushed. SSL will queue user data if the send path is not triggered up when socket option TCP_SSL_SEND_MIN_SIZE is greater than zero. All data will be sent out rigth away, if this function is called.

```
int SSL_SendFlush(int sd, int *error);
```

## Parameters

*sd*

Socket descriptor

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: Success
-1: failed

## See Also

**setsockopt**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|

| V2.02 |
| --- |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSL_ConnClose

Close a SSL connection. The user data will be flushed and a close notify message will be send to the SSL peer. User may also close the whole socket instead of only closing the SSL layer.

```
int SSL_ConnClose(int sd, int *error);
```

## Parameters

*sd*

Socket descriptor

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: Success
-1: failed

## Comments

Note that the peer may not accept any data after the SSL layer is closed.

## See Also

**closesocket**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.91 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.02 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SSL_GetConnState

Get status data of the specified SSL socket connection

```
int SSL_GetConnState(int sd, void far * * connStatePtr, int *error);
```

## Parameters

*sd*

Socket descriptor

*error*

Output parameter:  Failure **code**, 0 on success.

*connStatePtr*

Output parameter: Points to internal SSL connection data.

## Return Value

0: success, *connStatePtr points to the internal SSL socket connection data
-1: failed

## Comments

Currently this function is only required for the driver of the wireless module WL01. The driver requires the internal informations, when using the EAP_TLS protocol.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.17 | V1.17 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.12 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Crypt_MD5_Init

Init MD5 checksum generator.

```
int Crypt_MD5_Init ( Crypt_MD5_ContextPtr md5context );
```

## Parameters

*md5context*

Pointer to **MD5 context structure**

## Return Value

0: Success
-1: failed

## See Also

**Crypt_MD5_Update** **Crypt_MD5_Final**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index




BECK
THE POINT OF SYNERGY

# @CHIP-RTOS C Library - TCP/IP API

---

## Crypt_MD5_Update

Update MD5 checksum generator.

```
int Crypt_MD5_Update ( Crypt_MD5_ContextPtr md5context,
                       unsigned char far* inBuffer,
                       unsigned int length );
```

## Parameters

*md5context*

Pointer to **MD5 context structure**

*inBuffer*

Pointer to user data

*length*

Length of inBuffer in bytes

## Return Value

0: Success
-1: failed

## See Also

**Crypt_MD5_Init** **Crypt_MD5_Final**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Crypt_MD5_Final

Final MD5 checksum generator.

```
int Crypt_MD5_Final ( Crypt_MD5_ContextPtr md5context );
```

## Parameters

*md5context*

Pointer to **MD5 context structure**

## Return Value

0: Success
-1: failed

## Comments

After this call md5context->digest will hold the final hash value.

## See Also

**Crypt_MD5_Init** **Crypt_MD5_Update**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Crypt_SHA1_Init

Init SHA1 checksum generator.

```
int Crypt_SHA1_Init ( Crypt_SHA1_ContextPtr sha1context );
```

## Parameters

*sha1context*

Pointer to **SHA1 context structure**

## Return Value

0: Success
-1: failed

## See Also

**Crypt_SHA1_Update** **Crypt_SHA1_Final**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.08 |

## Crypt_SHA1_Update

Update SHA1 checksum generator.

```
int Crypt_SHA1_Update ( Crypt_SHA1_ContextPtr sha1context,
                        unsigned char far* inBuffer,
                        unsigned int length );
```

## Parameters

*sha1context*

Pointer to **SHA1 context structure**

*inBuffer*

Pointer to user data

*length*

Length of inBuffer in bytes

## Return Value

0: Success
-1: failed

## See Also

**Crypt_SHA1_Init** **Crypt_SHA1_Final**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Crypt_SHA1_Final

Final SHA1 checksum generator.

```
int Crypt_SHA1_Final ( Crypt_SHA1_ContextPtr sha1context );
```

## Parameters

*sha1context*

Pointer to **SHA1 context structure**

## Return Value

0: Success
-1: failed

## Comments

After this call sha1context->digest will hold the final hash value.

## See Also

**Crypt_SHA1_Init** **Crypt_SHA1_Update**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Crypt_AES_MakeEncryptKey

Make AES Encrypt Key.

```
int Crypt_AES_MakeEncryptKey ( Crypt_GenericKeyPtr key,
                               Crypt_AES_ContextPtr aescontext );
```

## Parameters

*key*

Pointer to **Generic key structure**

*aescontext*

Pointer to **AES context structure**

## Return Value

0: Success
-1: failed

## See Also

**Crypt_AES_MakeDecryptKey** **Crypt_AES_Encrypt** **Crypt_AES_Decrypt**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Crypt_AES_MakeDecryptKey

Make AES Decrypt Key.

```
int Crypt_AES_MakeDecryptKey ( Crypt_GenericKeyPtr key,
                               Crypt_AES_ContextPtr aescontext );
```

## Parameters

*key*

Pointer to **Generic key structure**

*aescontext*

Pointer to **AES context structure**

## Return Value

0: Success
-1: failed

## See Also

**Crypt_AES_MakeEncryptKey** **Crypt_AES_Encrypt** **Crypt_AES_Decrypt**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Crypt_AES_Encrypt

Encrypt AES message.

```
int Crypt_AES_Encrypt ( Crypt_AES_ContextPtr aescontext,
                        unsigned char far * in,                          unsigned
char far * out );
```

## Parameters

*aescontext*

Pointer to **AES context structure**

*in*

Pointer to input data

*out*

Pointer to output data

## Return Value

0: Success
-1: failed

## See Also

**Crypt_AES_MakeEncryptKey** **Crypt_AES_MakeDecryptKey** **Crypt_AES_Decrypt**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Crypt_AES_Decrypt

Decrypt AES message.

```
int Crypt_AES_Decrypt ( Crypt_AES_ContextPtr aescontext,
                        unsigned char far * in,                          unsigned
char far * out );
```

## Parameters

*aescontext*

Pointer to **AES context structure**

*in*

Pointer to input data

*out*

Pointer to output data

## Return Value

0: Success
-1: failed

## See Also

**Crypt_AES_MakeEncryptKey** **Crypt_AES_MakeDecryptKey** **Crypt_AES_Encrypt**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.11 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
| --- |
| V2.08 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Client_Installed

Check if PPP client is installed.

```
int PPP_Client_Installed ( void );
```

## Parameters

-- none --

## Return Value

0: PPP client is not installed
non-zero: PPP client is installed

## Comments

Tests if PPP client services are available in @CHIP-RTOS version and not disabled via CHIP.INI **configuration**.

## See Also

**PPP_Client_Open** **Get_Installed_Servers**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## PPP_Client_Open

Open a PPP connection.

```
int PPP_Client_Open ( PPPClient_Init far *ptr, int *error );
```

## Parameters

*ptr*

Pointer to a **PPPClient_Init** type data structure (declared in tcpipapi.h)

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

On success, the structure at `ptr` contains the needed IP data for further TCP/IP socket communication.

Refer to the PPPCLIE.C example for how to use this API. Also refer to the **PPPClient_Init** data structure documentation.

Note: **Only one PPP client connection can be opened at a time!!**

## SC1x3/SC23 Comments

This API applies to IPv4 type PPP connections. For IPv6 connections, see **PPP_Client_Open_IPv6** API.

## See Also

**PPP_Client_Installed** **PPP_Client_Close** **PPP_Client_GetStatus**
**PPP_Client_Open_IPv6**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Client_Open_IPv6

Open a PPP connection with IPv6 address configuration.

```
int PPP_Client_Open_IPv6 ( PPPClient_Init_IPv6 far *ptr,
                           int *error );
```

## Parameters

*ptr*

Pointer to a **PPPClient_Init_IPv6** type data structure (declared in tcpipapi.h)

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

IPv6 address configuration is done by an auto negotiation process. At connection time the peers negotiate an Interface-ID for the PPP device and auto configure their link-local scope addresses. After sucessful return of this function the `PPPClieipAddrStr` and `PPPClieRemipAddrStr` members of the `PPPClient_Init_IPv6` data structure are filled with the negotiated IPv6 addresses.

On success, the structure at `ptr` contains the needed IP data for further TCP/IP socket communication.

Refer to the PPP6CLIE.C example for how to use this API. Also refer to **PPPClient_Init_IPv6** data structure documentation.

Note: **Only one PPP client connection can be opened at a time!!**

## See Also

**PPP_Client_Installed** **PPP_Client_Close** **PPP_Client_GetStatus**
**PPP_Client_Open**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Client_Close

Close a PPP client connection.

```
int PPP_Client_Close ( int * error );
```

## Parameters

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

Within this API call, after closing the PPP session the PPP Client executes the modem hang-up commands specified in the **PPPClient_Init** data structure defined in the **PPP_Client_Open** call.

## See Also

**PPP_Client_Open**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## PPP_Client_GetStatus

Get PPP client status.

```
int PPP_Client_GetStatus ( int *error );
```

## Parameters

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

non-negative: PPP client **status** code
negative: Failure, PPP client **error** code

## Comments

This API function returns -2 if the version of RTOS does not support the PPP client services.

## See Also

**PPP_Client_Open** **PPP_Client_Installed**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Client_Get_DNSIP

Get DNS IP.

```
int PPP_Client_Get_DNSIP ( unsigned long far *IPaddress,
                           int primary_sec, int *error );
```

## Parameters

*IPaddress*

Output parameter: DNS address is stored here.

*primary_sec*

1: Get primary DNS address
2: Get secondary address.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

This API function reports the DNS addresses as negotiated by the remote PPP server.

## See Also

**PPP_Client_Open**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.04 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Client_SetOptions

Set PPP options for the PPP client.

```
int PPP_Client_SetOptions ( const PPP_Option far *ptr );
```

## Parameters

*ptr*

Pointer to an array of **PPP_Option** type data structures

## Return Value

-3 : PPP Connection already established
-2 : PPP client is not installed
-1 : Invalid Option(s)
0 : Success

## Comments

If you want to use this function, you have to call it before opening a connection. **Setting options during an established connection does not work!** The settings are only valid for the next established connection. While establishing a connection the `PPP_Option` structure referenced by the `ptr` parameter must remain valid. After the **PPP_Client_Open** function is finished, the structure can then be released.

The array of data structures at `ptr` is terminated by the first structure encountered with a NULL `optionValuePtr` member.

```
// Simplified C-Example:
unsigned int  ipcp_comp   = 1;
unsigned long DNS_Pri_IP  = 0L;
unsigned long DNS_Sec_IP  = 0L;

PPP_Option My_Options[] = {

  // Allow remote to peer to use VJ TCPIP header compression.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_ALLOW, PPP_IPCP_COMP_PROTOCOL,
      (const char *)&ipcp_comp, sizeof(ipcp_comp)},

  // Allow remote to peer to set primary DNS IP.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_PRI,
      (const char *)&DNS_Pri_IP, sizeof(DNS_Pri_IP)},

  // Allow remote to peer to set secondary DNS IP.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_WANT, PPP_IPCP_DNS_SEC,
```

```
      (const char *)&DNS_Sec_IP, sizeof(DNS_Sec_IP)},

  // Terminate list of options.
  { 0, 0, 0, NULL, 0}
} ;

//***** call the functions like this ****

// Install option with CLIB function
PPP_Client_SetOptions(&My_Options[0]);  // Point to first member of array

PPP_Client_Open(&pppclient);           // Open the connection
```

## See Also

PPP_Client_Open

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---

## PPP_Server_Installed

Check if PPP server is installed.

```
int PPP_Server_Installed ( void );
```

## Parameters

-- none --

## Return Value

zero: PPP server is not installed
non-zero: PPP server is installed

## Comments

The CHIP.INI can specify that the PPP server not be **installed**.

## See Also

**PPP_Server_Activate** **Get_Installed_Servers**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## PPP_Server_Suspend

Suspend the PPP server task.

```
int PPP_Server_Suspend (int timeoutsecs, int *error );
```

## Parameters

*timeoutsecs*

Time-out in seconds.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

Note that the time-out value in `timeoutsecs` depends on your time-out entries for the modem commands specified in chip.ini. If this call returns with -1, the most likely reason is that the modem commands are not finished after specified time-out.

## See Also

**PPP_Server_Activate** **PPP_Server_GetStatus**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

Chip.ini **entry:** PPPSERVER HANGUPTIMEOUTx time-out in seconds for wait on answer from modem

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Server_Activate

Activate the PPP server.

```
int PPP_Server_Activate ( int timeoutsecs, int *error );
```

## Parameters

*timeoutsecs*

Time-out in seconds.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

If successful, the PPP server is now able to serve a connection.

Note that the time-out value in `timeoutsecs` depends on your time-out entries for the modem commands specified in chip.ini. If this call returns with -1, the most likely reason is that the modem commands are not finished after specified time-out.

## See Also

**PPP_Server_Suspend** **PPP_Server_Installed** **PPP_Server_GetStatus**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Related Topics

Chip.ini **entry:** PPPSERVER HANGUPTIMEOUTx time-out in seconds for wait on answer from modem

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Server_GetStatus

Get the current state of the PPP server.

```
int PPP_Server_GetStatus ( void );
```

## Parameters

-- none --

## Return Value

PPP server state defined below

## Comments

PPP server states:

-2 PPP server is not installed.
-1 Error state, should not happen
1 Server disabled
2 Server enabled, waiting for connection
3 PPP connection is established
4 Server tries to hang up modem
5 Server tries to initialize modem

## See Also

PPP_Server_Suspend  PPP_Server_Activate  PPP_Server_Installed

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Server_Get_Cfg

Get the current main configuration data of the PPP server .

```
int PPP_Server_Get_Cfg ( PPP_IPCfg_Data far *ptr, int *error );
```

## Parameters

*ptr*

Output Parameter: Pointer to a **PPP_IPCfg_Data** type data structure which will be set by this function.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

Non-zero = Failure (see `error` output parameter)
zero: The user structure `PPP_IPCfg_Data` is filled with the PPP server configuration data

## Comments

This API applies to PPP server operating with IPv4 protocol. Refer to **PPP_Server_Get_Cfg_IPv6** API for IPv6 PPP server configuration access.

## See Also

**PPP_Server_Suspend** **PPP_Server_GetStatus** **PPP_Server_Activate**
**PPP_Server_SetOptions** **PPP_Server_Get_Cfg_IPv6**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Server_Get_Cfg_IPv6

Get the current main configuration data of the PPP server.

```
int PPP_Server_Get_Cfg_IPv6 (
                      PPP_IPCfg_Data_IPv6 far *ptr,
                      int *error );
```

## Parameters

*ptr*

Output parameter: Pointer to a **PPP_IPCfg_Data_IPv6** type data structure which will be set by this function.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

Non-zero = Failure (see `error` output parameter)
zero: The user structure `PPP_IPCfg_Data_IPv6` is filled with the PPP server configuration data

## Comments

Only PPP IPv6 connections are supported by this API. IPv6 address configuration is done by an auto negotiation process. At connection time, the peers negotiate an Interface-ID for the PPP device and auto configure their link-local scope addresses.

Refer to **PPP_Server_Get_Cfg** API for IPv4 PPP server configuration access.

## See Also

**PPP_Server_Suspend** **PPP_Server_GetStatus** **PPP_Server_Activate**
**PPP_Server_SetOptions** **PPP_Server_Get_Cfg**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## PPP_Server_SetOptions

Set PPP options for the PPP server.

```
int PPP_Server_SetOptions ( const PPP_Option far *ptr );
```

## Parameters

*ptr*

Pointer to an array of `PPP_Option` **type** data structures (defined in tcpipapi.h).

## Return Value

-3: PPP Connection already established
-2: PPP server is not installed
-1: Invalid Option(s)
0: Success

## Comments

It is only possible to set PPP options when the PPP server is suspended. (PPP server status reported by **PPP_Server_GetStatus** must be 1.) Setting options when the PPP server is active (states 2, 3, 4 and 5) does not have any effect. The installed options take affect when the PPP server is activated. If you want to reset the options, call this function with a null `ptr` parameter.

The array of data structures at `ptr` is terminated by the first structure encountered with a NULL `optionValuePtr` member.

```
//  Simplified C-Example:
unsigned int  ipcp_comp   = 1;
unsigned long DNS_Pri_IP  = 0L;
unsigned long DNS_Sec_IP  = 0L;

static PPP_Option My_Options[] = {
  // Allow remote to peer to use VJ TCPIP header compression.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_ALLOW, PPP_IPCP_COMP_PROTOCOL,
     (const char *)&ipcp_comp, sizeof(ipcp_comp)},

  // Allow remote to peer to set primary DNS IP, but do no request for it.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_ALLOW, PPP_IPCP_DNS_PRI,
     (const char *)&DNS_Pri_IP, sizeof(DNS_Pri_IP)},

  // Allow remote to peer to set secondary DNS IP, but do no request for it.
  { PPP_IPCP_PROTOCOL, PPP_OPTION_ALLOW, PPP_IPCP_DNS_SEC,
     (const char *)&DNS_Sec_IP, sizeof(DNS_Sec_IP)},

  { 0, 0, 0, NULL, 0}   // End of array marker
```

```
} ;

//*****call the functions like this****

PPP_Server_Suspend(20);                 // Suspend the PPP server
// Install option with CLIB function
PPP_Server_SetOptions(&My_Options[0]);  // point to first member of array

PPP_Server_Activate(20);                // Activate
```

## See Also

PPP_Server_Suspend  PPP_Server_GetStatus  PPP_Server_Activate
PPP_Server_Get_Cfg

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.03 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## Get_FTP_Login_Counters

Access the FTP server login counters.

```
int Get_FTP_Login_Counters (
                    unsigned long far * * FTP_Login_Count,
                    unsigned long far * * FTP_Login_failCount );
```

## Parameters

*FTP_Login_Count*

Output parameter: A far pointer to the 32 bit login counter will be output to the location pointed to by this parameter. On failure, a NULL pointer is output here.

*FTP_Login_failCount*

Output parameter: A far pointer to the 32 bit login fail counter will be output to the location pointed to by this parameter. On failure, a NULL pointer is output here.

## Return Value

0: success
non-zero: @CHIP-RTOS doesn't support FTP server pointers are set to NULL).

## See Also

Get_Telnet_Login_Counters

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

End of document

## Get_Telnet_Login_Counters

Access the Telnet server login counters.

```
int Get_Telnet_Login_Counters(
                unsigned long far * * Telnet_Login_Count,
                unsigned long far * * Telnet_Login_failCount);
```

## Parameters

*Telnet_Login_Count*

Output parameter: A far pointer to the 32 bit login counter will be output to the location pointed to by this parameter. On failure, a NULL pointer is output here.

*Telnet_Login_failCount*

Output parameter: A far pointer to the 32 bit login fail counter will be output to the location pointed to by this parameter. On failure, a NULL pointer is output here.

## Return Value

0: success
non-zero: @CHIP-RTOS doesn't support Telnet server (pointers are set to NULL).

## See Also

Get_FTP_Login_Counters

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

End of document

## Get_Telnet_State

Check if have an active telnet session.

```
int Get_Telnet_State ( int *error );
```

## Parameters

*error*

Output parameter: 0: no Telnet session
1: Telnet session is active
-1: BIOS doesn't support Telnet server

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## See Also

Get_Telnet_Login_Counters  Get_Installed_Servers

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Get_Installed_Servers

Get information about running servers and interfaces of the TCP/IP Stack.

```
void Get_Installed_Servers ( unsigned int far* first,
                             unsigned int far* second );
```

## Parameters

*first*

The first eight possible available servers:
- Bit 0: GIS_ETHERNET - Ethernet device
- Bit 1: GIS_PPP_SERVER - PPP server
- Bit 2: GIS_PPP_CLIENT - PPP client
- Bit 3: GIS_WEB_SERVER - Web server
- Bit 4: GIS_TELNET_SERVER - Telnet server
- Bit 5: GIS_FTP_SERVER - FTP server
- Bit 6: GIS_TFTP_SERVER - TFTP server
- Bit 7: GIS_DHCP - DHCP client

*second*

The next eight possible available servers:
- Bit 0: GIS_SNMP_MIB - SNMP MIB variables support
- Bit 1: GIS_CFG_SERVER - UDP Config server
- Bit 2: GIS_PING_CLIENT - Ping client

## Return Value

-- none --

## Comments

The masked listed here are defined in the tcpip.h header file.

## See Also

PPP_Server_Installed  PPP_Client_Installed

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Get_TCPIP_Memory_Status

Get information about TCP/IP stack memory usage.

```
void Get_TCPIP_Memory_Status ( unsigned long far *total,
                               unsigned long far *used );
```

## Parameters

*total*

Output parameter: Pointer to 32 bit location what will receive the maximum number of bytes of available memory.

*used*

Output parameter: Pointer to 32 bit location what will receive the number of bytes of memory currently used.

## Return Value

-- none --

## See Also

Get_SNMP_Data  Get_TCPIP_Statistics

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

## Get_TCPIP_Statistics

Access the internal network packet counters.

```
void Get_TCPIP_Statistics ( Packet_Count far * *pkt_cnt_ptr );
```

## Parameters

*pkt_cnt_ptr*

Output Parameter: Pointer to location in user memory which will receive a pointer to a **Packet_Count** data structure residing in @CHIP-RTOS memory space (read-only!). This referenced @CHIP-RTOS data structure contains further pointers to the counters for each protocol.

## Return Value

-- none --

## Comments

The counters referenced by `Packet_Count` data structure members `count_all_packets` and `cnt_all_sended_packets` count only Ethernet packets.

Other counters referenced also count the packets from and to other devices: e.g. local loop back packets and PPP packets.

The user is free to read and/or reset any of these referenced counters.

## See Also

**Get_TCPIP_Memory_Status** **Get_SNMP_Data**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Get_SNMP_Data

Get internal TCP/IP SNMP variables.

```
int Get_SNMP_Data ( unsigned char which,
                    void far * *snmp_mib_ptr );
```

## Parameters

*which*

Specifies which data to access.

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

*snmp_mib_ptr*

Output Parameter: Pointer to a far pointer in user memory which will be set with a pointer to requested object time within the @CHIP-RTOS memory space.

## Return Value

non-zero : BIOS without internal SNMP mib variables
0: Location refereced by `snmp_mib_ptr` contains a pointer to the structure

## Comments

This API provides access to the internal TCP/IP variables which will be required if the user is to implement a SNMP (Simple Network Management Protocol) agent.

## SC1x3/SC23 Comments

This API currently only supports IPv4 protocol.

## SC1x Comments

These structures are only available in @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS.

The SNMP variables are not a part of our current official 6 SC1x @CHIP-RTOS versions. You must order directly an SC1x @CHIP-RTOS version which includes this feature.

## See Also

Get_TCPIP_Memory_Status  Get_TCPIP_Statistics  Get_SNMP_DataEx

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Get_SNMP_DataEx

Get internal TCP/IP SNMP variables (Extended).

```
int Get_SNMP_DataEx ( unsigned char which,
                      void far * *snmp_mib_ptr,
                      int *max );
```

## Parameters

*which*

Specifies which data to access.

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
7: Set array of pointers to **IfMib** entries of all current open TCP/IP device interfaces (Local Loopback, Ethernet, PPP server, PPP client). The value at `max` must specify the maximum number of pointers in this array.
8: Get pointer to unsigned long *system up-time* variable (10 Hz counter).

*snmp_mib_ptr*

Output Parameter:

For cases where `which` is not set to 7, this is a pointer to a single far pointer in user memory which will be set by this API with a pointer to requested object within the @CHIP-RTOS memory space.

For case where `which` is set to 7, this is a pointer to an array of far pointers within the caller's memory. This array will be set by this API call with pointers to **IfMib** data structures located in @CHIP-RTOS memory. One pointer for each device interface will be reported, up to a maximum limited by the count specified at the `max` input parameter.

*max*

Input/Output Parameter: This parameter only applies to case where `which` is set to 7. It is otherwise not used. For `which` case 7, this location must be preset by caller to the number of pointers in the array referenced by the `snmp_mib_ptr` input parameter. On return, the referenced count will indicate the number of devices for which **IfMib** pointers have been listed in the `snmp_mib_ptr` output array.

## Return Value

non-zero : RTOS version without internal SNMP mib variables
0: Location refereced by `snmp_mib_ptr` contains a pointer to the structure

## Comments

This API provides access to the internal TCP/IP variables which will be required if the user is to implement a SNMP (Simple Network Management Protocol) agent.

## SC1x3/SC23 Comments

This API currently only supports IPv4 protocol.

## SC1x Comments

These structures are only available in @CHIP-RTOS versions which contain the SNMP option. A SNMP agent is not part of the @CHIP-RTOS.

The SNMP variables are not a part of our current official 6 SC1x @CHIP-RTOS versions. You must order directly an SC1x @CHIP-RTOS version which includes this feature.

## See Also

**Get_TCPIP_Memory_Status** **Get_TCPIP_Statistics** **Get_SNMP_Data**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.01 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## DHCP_use

Set/Reset DHCP usage on the Ethernet interface.

```
int DHCP_use ( unsigned char dhcp_use );
```

## Parameters

*dhcp_use*

0: do not use DHCP, 1: Use DHCP.

## Return Value

0: success
non-zero: DHCP not available in this RTOS version.

## Comments

This entry becomes valid only after rebooting the system or after calling the **Reconfigure_ethernet** library function.

If DHCP is changed from 1 to 0 then a new IP address, subnet mask and gateway should be set with either the prompt **commands** or the **Set_IPConfig** library call.

## See Also

**Reconfigure_ethernet** **Set_IPConfig** **Get_DHCP_Status**
**Get_DHCP_Status_Ext**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Get_DHCP_Status

Get DHCP status of the Ethernet interface.

```
void Get_DHCP_Status ( unsigned int *dhcp_use,
                       unsigned int *dhcp_stat );
```

## Parameters

*dhcp_use*

Output parameter: Pointer to a where a "DHCP in use" boolean value will be output to. The location will be set to '1' if the DHCP is currently being used and '0' if not in use.

*dhcp_stat*

Output parameter: Pointer to a where a DHCP status word will be stored. The status is encoded as follows:

0: System is not configured (is in progress)
1: System is configured by a DHCP Server
2: System configure retry failed (or no retry started before)

## Return Value

-- none --

## See Also

Get_DHCP_Status_Ext  DHCP_use

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Get_DHCP_Status_Ext

Get DHCP status of the Ethernet interface.

```
void Get_DHCP_Status_Ext ( unsigned int *dhcp_use,
                           unsigned int *dhcp_stat,
                           UserEthDhcp_Entry far * *dhcpptr );
```

## Parameters

*dhcp_use*

Output parameter: Pointer to a where a "DHCP in use" boolean value will be output to. The location will be set to '1' if the DHCP is currently being used and '0' if not in use.

*dhcp_stat*

Output parameter: Pointer to a where a DHCP status word will be stored. The status is encoded as follows:

0: System is not configured (is in progress)
1: System is configured by a DHCP Server
2: System configure retry failed (or no retry started before)

*dhcpptr*

Output Parameter: Pointer to a pointer that will be set by this API call, referencing a **UserEthDhcp_Entry** data structure within @CHIP-RTOS memory. This structure will contain some DHCP configuration data (read-only!). If no data is available the pointer will be set to NULL.

## Return Value

-- none --

## Comments

The UserEthDhcp_Entry data is a read-only information. Do not write to this data structure!

## See Also

**Get_DHCP_Status** **DHCP_use**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Ping_Open

Open and start ICMP echo requests.

```
int Ping_Open ( Ping far *pingptr, int *error );
```

## Parameters

*pingptr*

Pointer to user's `Ping` **structure**.
The contents of this structure will be initialized by this function call.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

-2: Ping API not supported by RTOS version
-1: Failure, see reported `error` code.
>= 0: Socket descriptor

## See Also

**Ping_Close** **Ping_Statistics**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## Ping_Close

Stop ICMP echo requests.

```
int Ping_Close ( int sd );
```

## Parameters

*sd*

Socket descriptor.

## Return Value

-2: Ping API not supported by RTOS
-1: Ping close failed, should not happen, only if socket descriptor is invalid
0: Success

## See Also

Ping_Open  Ping_Statistics

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## Ping_Statistics

Retrieve ping information.

```
int Ping_Statistics ( Ping far *pingptr );
```

## Parameters

*pingptr*

Output parameter: Pointer to user's `Ping` **structure** from the **Ping_Open** call.

## Return Value

-2: Ping API not supported by RTOS version
-1: Failure
0: Success

## Comments

The structure at `pingptr` is filled with the ping statistics.

## See Also

**Ping_Open** **Ping_Close**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## Set_FTPServer_Idle_Timeout

Set FTP server time-out.

```
int Set_FTPServer_Idle_Timeout ( unsigned int timeout );
```

## Parameters

*timeout*

Time out value in seconds.

## Return Value

0: success
non-zero: FTP server not available in this RTOS version

## Comments

This function inserts the specified FTP time-out into the CHIP.INI file. A reboot is required, before the new time-out value takes affect.

Refer to **FTP_TIMEOUT** in CHIP.INI documentation.

## See Also

**Set_TelnetServer_Idle_Timeout** **Get_FTPServer_Idle_Timeout** **Get_TelnetServer_Idle_Timeout**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## Set_TelnetServer_Idle_Timeout

Set Telnet server time-out.

```
int Set_TelnetServer_Idle_Timeout ( unsigned int timeout );
```

## Parameters

*timeout*

Time out in seconds.

## Return Value

0: success
non-zero: Telnet server not available in this RTOS version

## Comments

This function inserts the specified Telnet time-out into the CHIP.INI file. The new time-out value takes affect immediately, without requiring a reboot.

Refer to **TELNET_TIMEOUT** in CHIP.INI documentation.

## See Also

**Set_FTPServer_Idle_Timeout** **Get_TelnetServer_Idle_Timeout** **Get_FTPServer_Idle_Timeout**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## Get_FTPServer_Idle_Timeout

Get FTP server time-out value.

```
unsigned int Get_FTPServer_Idle_Timeout ( void );
```

## Parameters

-- none --

## Return Value

0xFFFF or 0xFFFE: FTP server not available in this RTOS version
Otherwise: Existing FTP server time-out value, in seconds.

## Comments

This function accesses the FTP server's existing time-out value.

Refer to **FTP_TIMEOUT** in CHIP.INI documentation.

## See Also

**Set_FTPServer_Idle_Timeout** **Set_TelnetServer_Idle_Timeout** **Get_TelnetServer_Idle_Timeout**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Get_TelnetServer_Idle_Timeout

Get Telnet server time-out value.

```
unsigned int Get_TelnetServer_Idle_Timeout ( void );
```

## Parameters

-- none --.

## Return Value

0xFFFF or 0xFFFE: Telnet server not available in this RTOS version
Otherwise: Existing Telnet server time-out value, in seconds.

## Comments

This function accesses the Telnet server's existing time-out value.

Refer to **TELNET_TIMEOUT** in CHIP.INI documentation.

## See Also

**Set_TelnetServer_Idle_Timeout** **Set_FTPServer_Idle_Timeout** **Get_FTPServer_Idle_Timeout**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

## AddStaticRouteEx

Add a route for a interface (extended version).

```
int AddStaticRouteEx ( IP_INTERFACE which_interface,
                       const Route_Entry far *route,
                       DevUserIfaceHandle DevHandlePtr,
                       int *error);
```

## Parameters

*which_interface*

Enumeration type:

- II_ETHERNET = 0: Ethernet
- II_PPP_SERVER = 1: PPP server
- II_PPP_CLIENT = 2: PPP client
- II_USER_DEVICE = 3: User device

*route*

Pointer to user `Route_Entry` **structure**

*DevHandlePtr*

If `which_interface==3` : User driver **Device handle** (e.g. ETH1 or WLAN driver handle)

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## Comments

The `Route_Entry` structure is defined in tcpipapi.h:

Refer to this RTOS API function's **documentation** for more details.

## See Also

DelStaticRoute
AddDefaultGateway DelDefaultGateway GetDefaultGateway

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.15 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

End of document

## DelArpEntryByPhys

Delete ARP entry by physical address.

```
int DelArpEntryByPhys ( const unsigned char far *phy,
                        int *error );
```

## Parameters

*phy*

Pointer to the physical address (6 Byte)

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## Comments

Refer to this RTOS API function's **documentation** for more details.

## See Also

**AddArpEntry** **GetARPRouteCacheEntries**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## AddArpEntry

Add an entry in the ARP table.

```
int AddArpEntry ( const unsigned long far *ip,
                  const unsigned char far *phy,
                  int *error );
```

## Parameters

*ip*

Pointer to the IP address

*phy*

Pointer to the physical address (6 Byte)

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## See Also

**DelArpEntryByPhys** **GetARPRouteCacheEntries**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## GetARPRouteCacheEntries

This function returns all valid entries in the internal ARP/Route cache table.

```
int GetARPRouteCacheEntries(
                ArpRouteCacheEntry far * arpuserptr);
```

## Parameters

*arpuserptr*

Output Parameter: Pointer to a user provided array of 64 data **structures** for storing ARP/Route cache table.

## Return Value

Number of valid entries output to table at `arpuserptr`. This user provided array contains the current ARP/Route cache table.

## See Also

**DelArpEntryByPhys** **AddArpEntry**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Add_IGMP_Membership

Install an IP multicast address entry.

```
int Add_IGMP_Membership ( const unsigned long far *MulticastIP,
                          const unsigned char far *MacAddress,
                          int *error );
```

## Parameters

*MulticastIP*

Pointer to multicast IPv4 address in network byte order.

*MacAddress*

Pointer to a 6 byte array that contains the corresponding Ethernet address.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## Comments

IP Multicasting is the Internet abstraction of hardware multicasting. It allows transmission of IP datagrams to a group of hosts that form a single multicast group. Membership in a multicast group is dynamic. Hosts may join or leave the 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 binary 1110. IP multicast addresses range from 224.0.0.0 through 239.255.255.255.

For the usage of IP multicasting on 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. In this mode it receives any incoming IP packet with the mapped Ethernet multicast address and forwards it to the TCP/IP layer. Each IP multicast packet will be sent with the mapped Ethernet multicast address.

Because of the feature, that a multicast IP packet will be received by any member of a multicast group, sending and receiving of IP multicast packets is only usable with UDP sockets (datagram sockets).

After installing a IP multicast address with this API, the application programmer is able use this address as a destination address when sending datagrams. A UDP socket is able to receive datagrams at the

specified multicast address.

The maximum number of supported IP multicast addresses in the IPC@CHIP® is limited to 15. Before installing an IP multicast address, you can find out the corresponding Ethernet multicast address with **IPMulticast_to_MacAddr** API.

This implementation does not support multicast routing. Sending and receiving multicast datagrams works only at local network.
Using multicast addresses is only possible on the Ethernet interface.

## SC1x3/SC23 Comments

This API applies only to IPv4 multicast.

## See Also

**Drop_IGMP_Membership** **IPMulticast_to_MacAddr**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Drop_IGMP_Membership

Delete an IP multicast address entry.

```
int Drop_IGMP_Membership ( const unsigned long far *MulticastIP,
                           int *error );
```

## Parameters

*MulticastIP*

Pointer to multicast IP address to be deleted. This address is in network byte order.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## SC1x3/SC23 Comments

This API applies only to IPv4 multicast.

## See Also

**Add_IGMP_Membership** **IPMulticast_to_MacAddr**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## IPMulticast_to_MacAddr

Map IP multicast address to Ethernet address.

```
int IPMulticast_to_MacAddr (
                    const unsigned long far *MulticastIP,
                    unsigned char far *MacAddress,
                    int *error );
```

## Parameters

*MulticastIP*

Pointer to multicast IP address in network byte order.

*MacAddress*

Output Parameter: Pointer to a 6 byte array where the corresponding MAC address will be stored by this function.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## 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 multicast 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

## SC1x3/SC23 Comments

This API applies only to IPv4 multicast addresses.

## See Also

Add_IGMP_Membership  Drop_IGMP_Membership

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.02 | V1.00 | V1.00 | V0.90 | V1.00 |

This API List
List of C Libraries
@CHIP-RTOS Main Index

---

End of document

---

## Dev_Open_Interface

Install user device driver

```
int Dev_Open_Interface(DevUserDriver far * DriverInfo,
                       int * errorcode);
```

## Parameters

*DriverInfo*

Pointer to **user driver information**.

*errorcode*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `errorcode` .

## See Also

**Dev_Close_Interface** **Dev_Recv_Interface** **Dev_Recv_Wait**
**Dev_Notify_ISR** **Dev_Get_Buffer** **Dev_Send_Complete**
**Dev_Wait_DHCP_Complete** **Dev_PPP_Register_Chap** **Dev_PPP_Register_Pap**
**Dev_PPP_Set_Option** **Dev_PPP_Get_Peer_IP** **Dev_PPP_Get_Peer_DNSIP**
**Dev_PPP_Get_Peer_IPv6**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Open_Interface_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Related Topics

**Example** - Installing device driver

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Close_Interface

Close user device driver/interface.

```
int Dev_Close_Interface(DevUserIfaceHandle DevHandlePtr,
                int * errorcode);
```

## Parameters

*DevHandlePtr*

**Device handle**

*errorcode*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `errorcode` .

## See Also

**Dev_Open_Interface**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Close_Interface_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Related Topics

**Example** - Closing device driver

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Recv_Interface

Receive and process incoming data at the TCP/IP stack.

```
int Dev_Recv_Interface(DevUserIfaceHandle DevHandlePtr,
                int * errorcode);
```

## Parameters

*DevHandlePtr*

Device handle

*errorcode*

Output parameter: Failure code, 0 on success.

## Return Value

0: success
-1: See reported `errorcode` .

## See Also

Dev_Open_Interface  Dev_Recv_Wait  Dev_Notify_ISR
Dev_Send_Complete  Dev_Wait_DHCP_Complete

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Recv_Interface_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Related Topics

**Example** - function which would be called by this API
**Example** - Receiver Task which calls this API

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Recv_Wait

Waits for data received signal from Interrupt Service Routine.

```
int Dev_Recv_Wait(DevUserIfaceHandle DevHandlePtr,
                  int * errorcode);
```

## Parameters

*DevHandlePtr*

Device handle

*errorcode*

Output parameter: Failure code, 0 on success.

## Return Value

0: success
-1: See reported `errorcode` .

## See Also

Dev_Open_Interface Dev_Recv_Interface Dev_Notify_ISR
Dev_Send_Complete Dev_Wait_DHCP_Complete

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Recv_Wait_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Related Topics

**Example** - Receiver Task

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Notify_ISR

Notify the TCP/IP stack from inside of an Interrupt Service Routine that there is incoming data available and/or that the device has sent a frame successfully.

```
int Dev_Notify_ISR( DevUserIfaceHandle DevHandlePtr,
                    unsigned int receivedPackets,
                    unsigned int sentPackets);
```

## Parameters

*DevHandlePtr*

Device handle

*receivedPackets*

Number of available received packets

*sentPackets*

Number of sent packets completed (provisional)

## Return Value

0: success
-1: See reported `errorcode` .

## See Also

Dev_Open_Interface  Dev_Recv_Wait

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Notify_ISR_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Related Topics

**Example** - User device ISR

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Get_Buffer

Get a buffer from TCP/IP stack's pre-allocated buffer for storing incoming data from this device.

```
int Dev_Get_Buffer( DevUserBufferHandle BufHandlePtr,
                    char far * far * buffer,
                    unsigned int length);
```

## Parameters

*BufHandlePtr*

Output Parameter: Pointer to location where this function will on success set a handle to the allocated buffer. This handle is used internally by the @Chip-RTOS TCP/IP stack to manage the subsequent release of this allocated buffer.

*buffer*

Output Parameter: Pointer to a variable of type **DevUserBufferHandle** where the user buffer handle is stored. (This value is for internal use.)

*length*

Number of bytes buffer required

## Return Value

0: success , *buffer points to the beginning of the allocated buffer
-1: Allocation failed, *buffer is a null pointer.

## See Also

**Dev_Open_Interface** **Dev_Recv_Interface** **Dev_Send_Complete**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Get_Buffer_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```
This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Related Topics

**Receiver** function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Send_Complete

This API signals the TCP/IP stack that the final frame in a message has been sent. (Note that a message can be fragmented into multiple link layer frames.)

```
int Dev_Send_Complete(DevUserIfaceHandle DevHandlePtr);
```

## Parameters

*DevHandlePtr*

**Device handle**

## Return Value

0: success
Otherwise: error

## See Also

**Dev_Open_Interface** **Dev_Recv_Interface** **Dev_Recv_Wait**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Send_Complete_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Related Topics

**Example** - User's Device Send Function

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Wait_DHCP_Complete

Waits for completion of IP configuration process by DHCP ("*Dynamic Host Configuration Protocol* ").

```
int Dev_Wait_DHCP_Complete(DevUserDriver far * DriverInfo,
                           unsigned int time_s,
                           int * errorcode);
```

## Parameters

*DriverInfo*

Pointer to **user driver information**.

*time_s*

Timeout seconds. This value will depend on the DHCP server used. A minimum value of 15 seconds is recommended.

*errorcode*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `errorcode` .

## See Also

**Dev_Open_Interface** **Dev_Recv_Interface** **Dev_Recv_Wait**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Wait_DHCP_Complete_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Related Topics

**Example** - User's receiver task

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_PPP_Register_Chap

Register a PPP CHAP authentication callback function.

```
int Dev_PPP_Register_Chap(DevUserIfaceHandle DevHandlePtr,
                          const void far * AuthCallbackFunction,
                          int *error);
```

## Parameters

*DevHandlePtr*

**Device handle**

*AuthCallbackFunction*

Pointer to the authentication callback function.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: failed

## Comments

The prototype of the user provided function is defined as:

```
char huge * myCHAPAuthenticate(char *username) ;
```

which returns the password (secret) corresponding to this user. If the user name is invalid, this function must return a NULL pointer.

## See Also

**Dev_PPP_Register_Pap** **Dev_PPP_Set_Option** **Dev_PPP_Get_Peer_IP**
**Dev_PPP_Get_Peer_DNSIP** **Dev_PPP_Get_Peer_IPv6**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_PPP_Register_Chap_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_PPP_Register_Pap

Register a PPP PAP authentication callback function.

```
int Dev_PPP_Register_Pap(DevUserIfaceHandle DevHandlePtr,
                         const void far * AuthCallbackFunction,
                         int *error);
```

## Parameters

*DevHandlePtr*

Device handle

*AuthCallbackFunction*

Pointer to the authentication callback function.

*error*

Output parameter: Failure code, 0 on success.

## Return Value

0: success
-1: failed

## Comments

The prototype of the user provided function is defined as:

```
int huge * myPAPAuthenticate(char *username, char * password) ;
```

which returns 1 for a valid username/password pair, else 0.

## See Also

Dev_PPP_Register_Chap  Dev_PPP_Set_Option  Dev_PPP_Get_Peer_IP
Dev_PPP_Get_Peer_DNSIP  Dev_PPP_Get_Peer_IPv6

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_PPP_Register_Pap_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_PPP_Set_Option

Set PPP option on an own PPP device driver.

```
int Dev_PPP_Set_Option(DevUserIfaceHandle DevHandlePtr,
                       const PPP_Option far * ppp_option,
                       int *error);
```

## Parameters

*DevHandlePtr*

**Device handle**

*ppp_option*

Pointer to the **PPP option**

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: failed

## See Also

**Dev_PPP_Register_Chap** **Dev_PPP_Register_Pap** **Dev_PPP_Get_Peer_IP**
**Dev_PPP_Get_Peer_DNSIP** **Dev_PPP_Get_Peer_IPv6**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_PPP_Set_Option_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Get_IP

Get user device IP address and netmask.

```
int Dev_Get_IP(DevUserIfaceHandle DevHandlePtr,
               unsigned long * ipaddr),
               unsigned long * netmask,
               int * errorcode);
```

## Parameters

*DevHandlePtr*

**Device handle**

*ipaddr*

Output parameter: Pointer to 32 bit location which will receive the IP address in big endian (network) byte order.

*netmask*

Output parameter: Pointer to 32 bit location which will receive the net mask in big endian (network) byte order.

*errorcode*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `errorcode` .

## See Also

**Dev_Open_Interface** **Dev_PPP_Get_Peer_IP** **Dev_PPP_Get_Peer_DNSIP** **Dev_PPP_Get_Peer_IPv6**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Get_IP_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_PPP_Get_Peer_IP

Get IP address of a PPP peer.

```
int Dev_PPP_Get_Peer_IP(DevUserIfaceHandle DevHandlePtr,
                        long far * Address,
                        int *error);
```

## Parameters

*DevHandlePtr*

**Device handle**

*Address*

Output parameter: Pointer to the storage where the peer's IPv4 address in network byte order will be written by this API.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success, address points to the IP of the PPP peer
-1: failed

## See Also

**Dev_PPP_Register_Chap** **Dev_PPP_Register_Pap** **Dev_PPP_Set_Option**
**Dev_PPP_Get_Peer_DNSIP** **Dev_PPP_Get_Peer_IPv6** **Dev_Get_IP**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_PPP_Get_Peer_IP_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
```

```
#include <clib.h>
```
This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_PPP_Get_Peer_IPv6

Get IPv6 address of a PPP peer

```
int Dev_PPP_Get_Peer_IPv6(DevUserIfaceHandle DevHandlePtr,
                          struct sockaddr_storage *Address,
                          int *error);
```

## Parameters

*DevHandlePtr*

**Device handle**

*Address*

Output parameter: Pointer to the storage where the peer's IPv6 address in network byte order will be written by this API.

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success, address points to the IPv6 address of the PPP peer
-1: failed

## See Also

**Dev_PPP_Register_Chap** **Dev_PPP_Register_Pap** **Dev_PPP_Set_Option**
**Dev_PPP_Get_Peer_IP** **Dev_PPP_Get_Peer_DNSIP** **Dev_Get_IP**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_PPP_Get_Peer_IPv6_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
```

```
#include <clib.h>
```
This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.10 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_PPP_Get_Peer_DNSIP

Get IP address of the DNS server from a PPP peer.

```
int Dev_PPP_Get_Peer_DNSIP(DevUserIfaceHandle DevHandlePtr,
                           long far * Address,
                           int flag,
                           int *error);
```

## Parameters

*DevHandlePtr*

**Device handle**

*Address*

Output parameter: Pointer to the storage where the DNS server's IPv4 address will be written by this API.

*flag*

1: get IP of primary DNS server, 2: Get IP of secondary DNS server

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success, address points to the IP of the DNS server
-1: failed

## See Also

**Dev_PPP_Register_Chap** **Dev_PPP_Register_Pap** **Dev_PPP_Set_Option**
**Dev_PPP_Get_Peer_IP** **Dev_PPP_Get_Peer_IPv6**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function

which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_PPP_Get_Peer_DNSIP_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Get_Handle_By_Name

Get the driver handle pointer by a given driver name

```
int Dev_Get_Handle_By_Name(const char * drivername,
                           DevUserIfaceHandle far * DevHandlePtr,
                           int *error);
```

## Parameters

*DevHandlePtr*

Contains the found **Device handle**

*drivername*

Driver name e.g. NE2000 (type RTOS command ipcfg)

*error*

Output parameter:  Failure **code**, 0 on success.

## Return Value

0: success, DevHandlePtr contains the device handle pointer
-1: failed

## Comments

This function delivers the required driver handle pointer, which is necessary for using **Dev_Config_Iface**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Get_Handle_By_Name_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily

switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_Config_Iface

IPv4 configuration of an interface

```
int Dev_Config_Iface(DevIpv4IfaceCfg far * devcfg, int *error);
```

## Parameters

*devcfg*

**DevIpv4IfaceCfg** filled by the caller. Contains the device handle,IPv4 configuration and the multihomeindex number.

*error*

Output parameter:  Failure **code**, 0 on success.

## Return Value

0: success,
-1: failed

## Comments

This functions provides the possibility to configure a device interface (e.g. the standard internal ethernet interface) with multiple IP addresses by enter a particular multihome index number. If an interface is already configured at the specified multihomeindex number, the user first has to unconfigure it with **Dev_UnConfig_Iface**. DHCP configuration of a device is only possible at multihomeindex 0.
It is recommended to use this function only for changing ip configuration of own device interfaces or for changing IP configuration of the internal ethernet interface for multihomeindex 1..3 (SC1x3/SC23 1 ..4).
Clibrary-based example:

```
DevUserIfaceHandle DevHandlePtr;
DevIpv4IfaceCfg  devcfg;
int error;
//Get the driver handle by name
if(-1==Dev_Get_Handle_By_Name("NE2000",&DevHandlePtr,&error))
{
  printf("\r\nError: Cannot detect driver handle, code %d",error);
  return 0;
}
//Set ip configuration
devcfg.IfaceHandle    = DevHandlePtr;
devcfg.flags          = 0;
devcfg.multiHomeIndex = 1;  //configure particular multihomeindex 1
inet_addr("192.168.200.59",&devcfg.IpAddr);
inet_addr("255.255.255.0"",&devcfg.Netmask)

if(-1==Dev_Config_Iface(&devcfg,&error))
{
     printf("Error %d\r\n",error);
}
```

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Config_Iface_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Dev_Config_Iface_IPv6

IPv6 configuration of an interface

```
int Dev_Config_Iface_IPv6(DevIpv6IfaceCfg far * devcfg, int *error);
```

## Parameters

*devcfg*

**DevIpv4IfaceCfg** filled by the caller. Contains the device handle,IPv6 configuration and the multihomeindex number.

*error*

Output parameter:  Failure **code**, 0 on success.

## Return Value

0: success,
-1: failed

## Comments

This functions provides the possibility to configure a device interface (e.g. the standard internal ethernet interface) with multiple IPv6 addresses by enter a particular multihome index number. If an interface is already configured at the specified multihomeindex number, the user first has to unconfigure it with **Dev_UnConfig_Iface**. DHCP configuration of a device is only possible at multihomeindex 0.
It is recommended to use this function only for changing ip configuration of own device interfaces or for changing IP configuration of the internal ethernet interface for multihomeindex 1..4.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_Config_Iface_IPv6_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily

switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| n/a | n/a | n/a | V1.05 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Dev_UnConfig_Iface

Remove IPv4 or IPv6 device configuration

```
int Dev_UnConfig_Iface(DevUserIfaceHandle DevHandlePtr,
                       unsigned int proto,unsigned char mHomeIndex,
                       int *error);
```

## Parameters

*DevHandlePtr*

**Device handle**

*proto*

Protocol family AF:INET or AF_INET6

*mHomeIndex*

Multi home index number of the device

*error*

Output parameter:  Failure **code**, 0 on success.

## Return Value

0: success,
-1: failed

## Comments

Remove an IP configuration from an interface for a particular multi home index

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

On SC1x3/SC23 (RTOS version >=1.13, CLIB version >=2.10) there's also a version of this function which uses dynamic linking instead of a software interrupt, which reduces the function's overhead and thus speeds up your application. This function's name is `Dev_UnConfig_Iface_Dyn`. You should however not use this function directly. Instead you should add the define `TCPIP_DYN_LINK_DEV` to your

program before including the CLIB's header file(s):

```
#define TCPIP_DYN_LINK_DEV
#include <clib.h>
```

This will map the original function's name to the dynamic linking version. This way you can also easily switch existing programs to dynamic linking.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.10 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RegisterCallbackFunction

Register a TCP user callback function.

```
int RegisterCallbackFunction ( int sd,
                               void far *funcptr,
                               int eventflagmask,
                               int *error );
```

## Parameters

*sd*

Socket descriptor.

*funcptr*

Pointer to user's callback function.

*eventflagmask*

Event flag mask bit field (see menu below)

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0 = success
Non-zero = Failure (see `error` output parameter)

## Comments

The events which trigger the callback are controlled with the `eventflagmask` input parameter. The set of possible event flags are (defined in TCPIPAPI.H):

- CB_CONNECT_COMPLT - connection complete
- CB_ACCEPT - remote has established a connection to our listening server
- CB_RECV - incoming data arrived
- CB_SEND_COMPLT - sending of data has been acknowledged by the peer
- CB_REMOTE_CLOSE - peer has shut down the connection
- CB_SOCKET_ERROR - an error occured on the connection
- CB_RESET - peer has sent a rest on the connection
- CB_CLOSE_COMPLT - close has been completed

These flags can be OR'ed together to form the mask for the set of events at which you want to be called.

The callback function should satisfy the following prototype (Borland C):

```
void huge socketCallBackFunc(int socketdescriptor,
                             int eventFlags) ;
```

The callback function's parameter `eventFlags` contains the event(s) that have occured. This bit field is encoded the same as the `eventflagmask` input parameter to this API function.

Note that most of the call back calls will be made in the context of the drivers receive task (e.g. task ETH0). Therefore processing should be kept at a minimum in the call back function. The user call back function should set a flag or increase a counter and signal the user application task.

A socket callback may be removed by calling this API function with both a null `funcptr` and zero `eventflagmask` . **Closing** the socket also deactivates the callback. Before exiting a program, any sockets which have callbacks installed must either be closed or have the callback removed. Otherwise a callback could be present which references a program no longer loaded into memory.

## See Also

**opensocket** **closesocket** **RegisterIPCallbackHandler**
**opensocket_ext** **Register_ARP_User_CB** **RegisterIPCallbackHandler_Snd**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RegisterIPCallbackHandler

Register an IP User callback handler.

```
int RegisterIPCallbackHandler ( void far *funcptr );
```

## Parameters

*funcptr*

Pointer to user's IP callback Handler.

## Return Value

0 = success

## Comments

This function allows the application programmer to install a ip packet filter callback function. The application programmer can implement a function fitting the following type definition:

```
typedef int (huge *IPSortOutFuncPtr)(
                IpUserCallbackInfo *ipInfo );
```

If a function of this type is installed by the user, the TCP/IP stack will call this function at any incoming IP packet. Inside of this function the user is able to check the given IP parameters and decide if the TCP/IP stack should process this packet or ignore it. If the callback function returns -1 the incoming packet will be ignored by the TCP/IP stack.

To uninstall the callback function, this API call must be called with a null pointer. ***Do not forget to uninstall the function if your application exits!***

## SC1x3/SC23 Comments

**Important:** The callback will only execute on incoming IPv4 packets.

## See Also

RegisterCallbackFunction  Register_ARP_User_CB  RegisterIPCallbackHandler_Snd

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## RegisterIPCallbackHandler_Snd

Register an IP user callback handler on outgoing packets.

```
int RegisterIPCallbackHandler_Snd ( void far *funcptr );
```

## Parameters

*funcptr*

Pointer to user's IP callback Handler.

## Return Value

0 = success

## Comments

This function allows the application programmer to install an IP packet filter callback function. The application programmer can implement a function fitting the following type definition:

```
typedef int (huge *IPCheckOutPacketFuncPtr)(
                IpUserCallbackInfo_t *ipInfo );
```

If a function of this type is installed by the user, the TCP/IP stack will call this function at any end IP packet. Inside of this function the user is able to check the given IP parameters and decide if the TCP/IP stack should send this packet or ignore it. If the callback function returns -1 the incoming packet will be ignored by the TCP/IP stack.

To uninstall the callback function, this API call must be called with a null pointer. ***Do not forget to uninstall the function if your application exits!***

## SC1x3/SC23 Comments

**Important:** The callback will only execute on incoming IPv4 packets.

## See Also

RegisterCallbackFunction RegisterIPCallbackHandler Register_ARP_User_CB

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's

**documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.11 | V1.01 | V1.01 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Register_ARP_User_CB

Register an ARP User callback handler.

```
void Register_ARP_User_CB(void far * funcPtr);
```

## Parameters

*funcptr*

Pointer to ARP callback function (or set to null to remove a previously installed callback)

## Return Value

returns nothing

## Comments

The application programmer can implement a function fitting the following type definition:

```
typedef int (huge *MyArpCallbackFuncPtr)(
                ArpUserCallbackInfo_t far *arpInfo );
```

If a function of this type is installed by the user, the TCP/IP stack will call this function for any incoming ARP packet. The input parameter to the callback function allows access to the ARP packet. Inside of this function the user is able to check the content of the incoming ARP packet and decide whether the TCP/IP stack should process this packet or ignore it. If the callback function returns -1, the incoming packet will be ignored by the TCP/IP stack.
If the size of the returned structure **ArpUserCallbackInfo_t** is equal or bigger 12, the structure now contains a **devicehandle_ptr** to determine the source device of the incoming packet for filtering.

For structure of an ARP packet see **ArpHeader**.

To remove the callback function, this function call must be called with a null pointer. ***Do not forget to uninstall the callback if your application exits!***

Refer to this RTOS API function's **documentation** for more details.

## See Also

**RegisterCallbackFunction** **RegisterIPCallbackHandler** **RegisterIPCallbackHandler_Snd**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## api_sleep

The calling task sleeps for specified number of milliseconds.

```
void api_sleep ( unsigned int howlong );
```

This function is **deprecated**! Use **RTX_Sleep_Time** instead.

## Parameters

*howlong*

Sleep period in milliseconds

## Return Value

-- none --

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Reconfigure_ethernet

Reconfigure Ethernet interface after changing the IP configuration.

```
int Reconfigure_ethernet ( void );
```

## Parameters

-- none --.

## Return Value

0: success
non-zero: error code

(Error code 237 indicates that an Ethernet interface configuration was already in progress.)

## Comments

A new IP configuration set with the prompt commands `ip`, `netmask` and `gateway` (or the corresponding library Set_IPConfig call) 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 Set_IPConfig library call.

## See Also

Set_IPConfig  DHCP_use

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.00 | V1.00 | V1.00 | V0.90 | V1.00 |

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Get_Iface_Entries

This function returns information about all installed TCP/IP device interfaces.

```
int Get_Iface_Entries(Iface_Entry far * iface_entry, int max);
```

This function is **deprecated**! Use Get_Iface_Entries_Ext instead.

## Parameters

*iface_entry*

Output Parameter: Pointer to a user provided array of Iface_Entry type data structures which will receive the information.

*max*

Number of data structures in array. Up to this number of device interfaces will be reported.

## Return Value

Number of valid entries in the array at `iface_entry` containing reported device interface information.

## SC1x3/SC23 Comments

This function works only for IPv4 device interfaces. The Get_IPv6_Iface_Entries API should be used to read out information for IPv6 interfaces.

## See Also

Get_IPv6_Iface_Entries

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's documentation for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.10 | V1.00 | V1.00 | V0.90 | V1.00 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

End of document

## Get_Iface_Entries_Ext

This function returns information about all installed TCP/IP device interfaces.

```
int Get_Iface_Entries_Ext(Iface_Entry_Ext far * iface_entry, int max);
```

## Parameters

*iface_entry*

Output Parameter: Pointer to a user provided array of **Iface_Entry_Ext** type data structures which will receive the information.

*max*

Number of data structures in array. Up to this number of device interfaces will be reported.

## Return Value

Number of valid entries in the array at `iface_entry` containing reported device interface information.

## Comments

This function is different from **GET_IFACE_ENTRIES**. It stores at the user provided array from type **structures** all ip configurations (IP multihoming on one interface) of each interface found, not only the first one.

## SC1x3/SC23 Comments

This function works only for IPv4 device interfaces. The **Get_IPv6_Iface_Entries** API should be used to read out information for IPv6 interfaces.

## See Also

**Get_IPv6_Iface_Entries**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|
| V1.20 | V1.20 | V1.20 | V1.10 | V1.00 |

## Supported by @CHIP-RTOS C Library since version

| CLIB |
|---|
| V2.04 |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## SetIfaceMTU

Set MTU (Maximum Transfer Unit) of a device interface.

```
int SetIfaceMTU ( IP_INTERFACE which_interface,
                  const void far *userIfaceHandle,
                  int mtu,
                  int *error );
```

## Parameters

*which_interface*

Enumeration type:

- II_ETHERNET = 0: Ethernet
- II_PPP_SERVER = 1: PPP server
- II_PPP_CLIENT = 2: PPP client
- II_USER_DEVICE = 3: user device driver

*userIfaceHandle*

If `which_interface` is 3, this parameter is a pointer to user device handle.

*mtu*

Maximum Transfer Unit

*error*

Output parameter: Failure **code**, 0 on success.

## Return Value

0: success
-1: See reported `error` code.

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **V1.11** | **V1.01** | **V1.01** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---

## Get_IPv6_Iface_Entries

This function returns information about all installed TCP/IP IPv6 device interfaces.

```
int Get_IPv6_Iface_Entries(
                        IfaceIPv6_Entry far * iface_ipv6_entries,
                        int max);
```

## Parameters

*iface_ipv6_entries*

Output Parameter: Pointer to a user provided array of **IfaceIPv6_Entry** type data structures which will receive the information.

*max*

Number of data structures in array. Up to this number of device interfaces will be reported.

## Return Value

Number of valid entries in the array at `iface_ipv6_entries` containing reported device interface information.

## Comments

This function works only for IPv6 device interfaces. The **Get_Iface_Entries** API should be used to read out information for IPv4 interfaces.

## See Also

**Get_Iface_Entries**

## RTOS API

This library function invokes a RTOS software interrupt. Refer to this RTOS API function's **documentation** for more details.

## Supported since or modified in @CHIP-RTOS version

| SC12 | SC13 | SC11 | SC1x3 | SC23 |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **n/a** | **n/a** | **n/a** | **V0.90** | **V1.00** |

This **API List**
List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS C Library - USB API

---

Header File: `#include <USB_API.H>`

This documentation describes the low-level software-interrupt-based API of the IPC@CHIP®RTOS dual-mode USB-driver. The SC1x3/SC23 disposes of a USB-controller that is compliant with the **USB specification version 1.1**
.

Though the USB-API hides much of the complexity of the USB, one should fly over the specification to get a grasp of how USB works.

We also recommend to take notice of the **Requirements and Recommendations for USB Products with Embedded Hosts and/or Multiple Receptacles**
.

The USB-driver can be used in Device mode to implement a USB device (resp. function) which will then be connected to e.g. a PC-Host; or it can be used in Host mode. In this case USB devices can be connected to the IPC@CHIP®handled by device-specific drivers that use the USB-API.

The RTOS USB-driver can operate in only one mode at a time. To implement a USB Function (Device) using this API, an additional circuit is needed, which manages attach- and speed-detection. A diagram is included in the hardware manual of the SC1x3/SC23 showing how to prepare your hardware to operate in both modes. Any PIO can be used to control this circuit. The API switches the PIO as needed.

Please note that there's no Hub support in Host mode and that there are some restrictions regarding data structures handled by the RTOS USB-driver necessary to keep the driver small and fast. These mainly concern Configurations, Interfaces and Alternate Settings. These restrictions should not constrain you much when implementing a USB-device and most devices connected to an IPC@CHIP®operating in Host mode will fulfil the conditions too.

- There can only be one Configuration. (It is uncommon to use more than one Configuration though the specification allows it.)
- There can be at most 10 Interfaces.
- Each interface can have maximal 12 alternate settings.
- Alternate settings in Device mode can only be used to alter the maximum packet size and the polling interval of its assigned endpoints. It is impossible to change e.g. the number of assigned endpoints. This would also be very uncommon. In Host mode a device may also have a different number of endpoints and different transfer types in Alternate Settings.

**Common** **Device mode only** **Host mode only**

## Common Functions

| | |
|---|---|
| usbSend | Send data |
| usbFreeReceiveBuffer | Hand a receive buffer back to the driver |
| usbWakeup | Wake up suspended nodes |
| usbSofNotify | Enable/disable notification of SOF events |

## Device mode only Functions

| | |
|---|---|
| usbDeviceInit | Initialise driver for Device mode |
| usbDeviceDeinit | Deinitialise Device mode driver |
| usbDeviceRegisterInterface | Register an interface |
| usbDeviceRegisterAltSetting | Register an Alternate Setting |
| usbDeviceGetEps | Get numbers of endpoints associated with an interface |
| usbDeviceConfigEp | Configure an endpoint |
| usbDeviceAddSpecificDesc | Add a class- or verndor-specific descriptor |
| usbDeviceStart | Start the Device mode driver |
| usbDeviceStop | Stop the Device mode driver |
| usbDeviceStallEp | Set STALL condition for an endpoint |
| usbDeviceEnableEp | Enable/disable OUT direction of an endpoint |

## Host mode only Functions

| | |
|---|---|
| usbHostInit | Initialise driver for Host mode |
| usbHostDeinit | Deinitialise Host mode driver |
| usbHostIssueToken | Issue token |
| usbHostClearQueue | Clear an endpoint's queues |
| usbHostReserveEp0 | Reserve endpoint 0 |
| usbHostReleaseEp0 | Release endpoint 0 |
| usbHostGetDeviceInfo | Get Device information |
| usbHostGetConfigInfo | Get Configuration information |
| usbHostGetInterfaceInfo | Get Interface information |
| usbHostGetEpInfo | Get Endpoint information |
| usbHostGetString | Get string |
| usbHostGetStatus | Get status |
| usbHostSetClearFeature | Set/clear feature |
| usbHostSetInterface | Set interface |

## Topics

USB API Data Structures
USB API Error Codes

List of **C Libraries**
**@CHIP-RTOS** Main Index

---

End of document

---




# @CHIP-RTOS - IP Security Known Problems



## IP Security Implementation Problems

The IP Security functionality offered in the @Chip-RTOS is still under development. Currently there are the following known issues concerning the implementation.

## General

- The @Chip-RTOS IP Security has not been tested with IPv6.
- The @Chip-RTOS IP Security will not operate in ESP tunnel mode with **NAT** devices between the IPC@CHIP® computer and Microsoft Windows PC, for versions of Windows XP that use the "draft 2 NAT-IKE" (true of all Windows XP versions observed at time of this writing, year 2007). However, the ESP transport mode does work with these "draft 2" implementations.

End of document