

## Documentation of Application Programming Interface of FutureOS

This file is a documentation of the part of the FutureOS API which can be permanently used. The main API entry points are located in every FutureOS ROM (in the address range of &FF00 - &FFFF); some are located in the system RAM (&B800 - &BFFF). All needed programming information is provided. All API entries / OS functions are described in the following way:

**1. Short description:** the function is described in brief.

**2. Label:** this LABEL is the name of the OS function used in the (Z80-) source code. Look at the most recent LABEL - library file #EQU-API.ENG (on disc) for the addresses of API entry points or OS functions. Please use the appropriate LABEL in your source code when using an OS function or system-variable! Never use the direct address! Addresses may change in further versions of FutureOS, but the features of OS functions and system-variables should stay the same in future.

**3. ROM-number:** provides the logical number(s) of the OS ROM(s) in which the corresponding OS function is located. More than one number is possible if dealing with API entries present in all OS ROMs. This logical OS ROM number (&A-&D) may not be the physical ROM number! Only in 256 KB ROM cards FutureOS usually has identical logical and physical ROM numbers (&0A, &0B, &0C and &0D).

**4. Start address:** gives you the entry address of the current API entry or OS function in it's ROM(s). If it's present in more than one ROM there may be multiple addresses provided. This address is the same you'll find in the file #EQU-API.ENG for its LABEL.

**5. Jump in conditions:** These are the conditions needed when your code jumps to the API entry or OS function. Z80 registers and RAM-variables must be loaded with the correct values. That's described here.

**6. Jump out conditions:** They describe Z80 registers and RAM-variables after returning to your program code. More often registers and flags of the Z80 provide information about the success of the OS function used.

**7. Manipulated:** all the manipulations which have been done by the OS function are given. Registers and RAM-variables are most interesting. But changes in memory or what ever else are described too.

**8. Description:** this is the complete description of the API entry, the OS function or the used OS function.

**9. Attention:** Critical details are described here. All the functions of FutureOS are trimmed to high-speed. Therefore you have to deal with them in a defined way. Important points are mentioned here.

Most of the following OS functions are useable constantly. Under FutureOS they're present permanently. They are either located in all OS ROMs or in the system RAM. Hence you don't have to bank or swap the upper ROM if you want to use them.

Most of this OS functions deal with ROM-banking or RTC management.

You want the latest version of this file?

Look the FutureOS homepage:

<http://www.FutureOS.de>

## **SELECT / BANK IN FutureOS ROM A, B, C OR D**

**Short description:** The ROM A, B, C or D of FutureOS will be banked in.

**Labels:** OSRON\_A, OSRON\_B, OSRON\_C, OSRON\_D

**ROM-numbers:** A, B, C and D (all OS ROMs)

**Start addresses:**

&FF22 (OSRON\_A), &FF58 (OSRON\_B)

&FF8E (OSRON\_C), &FFD6 (OSRON\_D)

**Jump in conditions:** None. The new ROM is selected by the used Label.

**Jump out conditions:** The corresponding FutureOS ROM (A, B, C or D) was banked in between &C000 and &FFFF.

**Manipulated:** Only the selected upper ROM at &C000

**Description:** The OS functions OSRON\_A, OSRON\_B, OSRON\_C and OSRON\_D are used to select the upper ROM. This can be FutureOS ROM A, B, C or D. The used label / used OS function will select which OS ROM will be banked in. This way no parameters need to be passed over to these OS functions. In addition no registers or flags will be changed.

The OS functions of FutureOS are separated into four ROMs (they are named A, B, C or D). Therefore the user is responsible to bank in the correct ROM before an OS function is called.

This can be done either by using OSRON\_A,B,C,D or by using the subsequent API entry points. Of course you can also select an OS ROM 'by hand', please refer to the Manual.

These OS functions need 17 us each, therefore they're quick.

**Attention:** The RAM variable AKT\_ROM will NOT be changed by these OS functions.

## API: JUMP TO AN OS FUNCTION IN FutureOS ROM A, B, C OR D

**Short description:** This API entry will bank in another FutureOS ROM, then it will jump to an OS function in the new ROM.

**Label:** ROM\_A, ROM\_B, ROM\_C, ROM\_D

**ROM-number(s):** A, B, C and D (all OS ROMs)

**Start addresses:** &FF00 (ROM\_A), &FF06 (ROM\_B)  
&FF0C (ROM\_C), &FF12 (ROM\_D)

**Jump in conditions:** HL = Target-address of OS function in the new ROM

**Jump out conditions:** The new FutureOS-ROM (A, B, C or D) is banked in between &C000 and &FFFF.

Anything else depends on the used OS function in the new ROM.

**Manipulated:** BC, a new FutureOS ROM is banked in. All other registers and RAM-variables may be changed by the used OS function.

**Description:** The API entries ROM\_A, ROM\_B, ROM\_C and ROM\_D are used to bank in one of the four OS ROMs ROM A, B, C or D. Then a jump is made to a target OS function the new ROM (address provided in register HL).

After the target OS function returns the new OS ROM will stay active.

The address of the OS function must be given in register HL. Therefore you can't use OS functions which expect data in the HL register of the Z80 (for such a case you must use one of the ROM\_?2? API entries. Here: ? = A, B, C or D, see later).

**Whatever** happens after jumping back depends on the used OS function.

The ROM-select and the jump to the target-OS function are accomplished in 8 µs. One can't do it faster ;-)

An example of using API entry ROM\_B:

```
JUMP_TO_ROM_B LD HL,TARGET    ;address of the OS function
...
...                ;prepare other registers
CALL ROM_B        ;banks ROM B in and jumps to
                  ;the target OS function (HL).
```

**Attention:** Please don't use one of the four OS functions ROM\_A...D if the target OS function in the new OS ROM needs values in the HL register. HL can't be used because HL is loaded with the address of the target OS function already.

The RAM-variable AKT\_ROM is NOT changed through these API entries.

## **API: CALL AN OS FUNCTION IN ROM B, C or D - JUMP BACK TO ROM A**

**Short description:** An OS function is called in ROM B, C or D. After its return FutureOS ROM A is banked in.

**Label:** ROM\_A2B, ROM\_A2C, ROM\_A2D

**ROM-number(s):** This API entry is present in all OS ROMs: A, B, C and D

**Start addresses:** &FF18 (ROM\_A2B), &FF2A (ROM\_A2C), &FF3C (ROM\_A2D)

**Jump in conditions:** IX = address of OS function in target ROM (B, C, D)  
Other parameters depend on the **used** OS function **itself**

**Jump out conditions:** FutureOS ROM A is selected. Other parameters depend on the called OS function.

**Manipulated:** depends on called OS function

**Description:** API entries ROM\_A2B, ROM\_A2C and ROM\_A2D allow the call of an OS function in ROM B, C or D (independent of the active ROM).

After the end of the called OS function ROM A is switched on (again).

The address of the OS function which should be called is given in register IX. Therefore the called OS function isn't able to use values given in IX.

The register contents depend only on the called OS function.

All OS functions of the type ROM\_?2? (? = A,B,C or D) need 34  $\mu$ s.

**Attention:** The called OS function isn't able to get parameters in IX.

## **API: CALL AN OS FUNCTION IN ROM A, C or D - JUMP BACK TO ROM B**

**Short description:** An OS function is called in ROM A, C or D. After its return FutureOS ROM B is banked in.

**Label:** ROM\_B2A, ROM\_B2C, ROM\_B2D

**ROM-number(s):** This API entry is present in all OS ROMs: A, B, C and D

**Start addresses:** &FF4E (ROM\_B2A), &FF60 (ROM\_B2C), &FF72 (ROM\_B2D)

**Jump in conditions:** IX = address of OS function in target ROM (A, C, D)  
Other parameters depend on the OS function.

**Jump out conditions:** FutureOS ROM B is selected. Other parameters depend on the called OS function.

**Manipulated:** depends on called OS function

**Description:** API entries ROM\_B2A, ROM\_B2C and ROM\_B2D allow the call of an OS function in ROM A, C or D (independent of the active ROM).

After the end of the called OS function ROM B is switched on (again).

The address of the OS function which should be called is given in register IX. The called OS function isn't allowed to use values given in IX.

The register contents depend only on the called OS function.

All OS functions of the type ROM\_?2? need exact 34  $\mu$ s.

**Attention:** The called OS function can't get parameters in IX.

## **API: CALL AN OS FUNCTION IN ROM A, B or D - JUMP BACK TO ROM C**

**Short description:** An OS function is called in ROM A, B or D. After its return FutureOS ROM C is banked in.

**Label:** ROM\_C2A, ROM\_C2B, ROM\_C2D

**ROM-number(s):** This API entry is present in all OS ROMs: A, B, C and D

**Start addresses:** &FF84 (ROM\_C2A), &FF96 (ROM\_C2B), &FFA8 (ROM\_C2D)

**Jump in conditions:** IX = address of OS function in target ROM (A, B, D)  
Other parameters depend on the OS function.

**Jump out conditions:** FutureOS ROM C is selected. Other parameters depend on the called OS function.

**Manipulated:** see called OS function.

**Description:** API entries ROM\_C2A, ROM\_C2B and ROM\_C2D allow the call of an OS function in ROM A, B or D (independent of the active ROM).

After the end of the called OS function ROM C is switched on (again).

The address of the OS function which should be called is given in

register IX. The called OS function isn't allowed to use values given in IX.

The register contents depend only on the called OS function.

All OS functions of the type ROM\_?2? need exact 34  $\mu$ s.

**Attention:** The called OS function can't get parameters in IX.

## **API: CALL AN OS FUNCTION IN ROM A, B or C - JUMP BACK TO ROM D**

**Short description:** An OS function is called in ROM A, B or C. After its return FutureOS ROM D is banked in.

**Label:** ROM\_D2A, ROM\_D2B, ROM\_D2C

**ROM-number(s):** This API entry is present in all OS ROMs: A, B, C and D

**Start addresses:** &FFBA (ROM\_D2A), &FFCC (ROM\_D2B), &FFDE (ROM\_D2C)

**Jump in conditions:** IX = address of OS function in target ROM (A, B, C).  
Other parameters depend on the OS function.

**Jump out conditions:** FutureOS ROM D is selected. Other parameters depend on the called OS function.

**Manipulated:** see called OS function.

**Description:** API entries ROM\_D2A, ROM\_D2B and ROM\_D2C allow the call of an OS function in ROM A, B or C (independent of the active ROM).

After the end of the called OS function ROM D is switched on (again).

The address of the OS function which should be called is given in register IX. The called OS function isn't allowed to use values given in IX.

The register contents depend only on the called OS function.

All OS functions of the type ROM\_?2? need exact 34  $\mu$ s.

**Attention:** The called OS function can't get parameters in IX.



## READ REAL-TIME-CLOCK (M4 card or Dobbertin RTC) DATA INTO RAM

**Short description:** Read date and time of the Real-Time-Clock (RTC) of the M4 expansion card or the Dobbertin / dxs SmartWatch into RAM.

**Label:** LUHR

**ROM-number:** This OS function is located in the system RAM of the OS!

**Start address:** &BA66 – in RAM!

**Jump in conditions:** The physical number of the actual selected FutureOS ROM must be stored in the RAM variable AKT\_ROM.

The OS variable UHR\_ROM must contain either the number of the M4-ROM or the Dobbertin / dxs SmartWatch. See file #OS-VAR.ENG.

**Jump out conditions:** Eight data bytes from the M4 card RTC or the Dobbertin / dxs RTC have been read into RAM at UHR\_00. The previously used OS ROM (stored in AKT\_ROM) is active again.

**Manipulated:** AF, BC, DE, HL and eight bytes at UHR\_00

**Description:** LUHR reads either the RTC data of the M4 card or the Dobbertin / dxs SmartWatch into the RAM at UHR\_00. Further information about these bytes is given in the file #OS-VAR.ENG.

When you call LUHR the RAM variable AKT\_ROM must contain the physical number of the currently active / calling OS ROM. Because after the return of LUHR this ROM is switched back on again.

You could read the logical number of the active OS ROM (A, B, C, and D) from address &C001. The value &0A means ROM A, &0B means ROM B, &0C means ROM C and &0D means ROM D. But what you need is the physical ROM-select!

The physical ROM-number (ROM-select) of the ROMs A, B, C or D can be read directly in every selected OS-ROM.

Physical ROM-number of ROM A can be read from &FF01.

Physical ROM-number of ROM B can be read from &FF07.

Physical ROM-number of ROM C can be read from &FF0D.

Physical ROM-number of ROM D can be read from &FF13.

Every ROM-Byte is followed by one Byte &DF. This allows the following code... (example to select OS ROM C).

```
LD BC,(&FF0D) ;physical number of ROM C
                ;is followed by &DF (in ROM)
OUT (C),C      ;switch ROM C on.
```

The Dobbertin / dxs RTC is connected to a CPC like any external ROM. To read data from the watch, the corresponding ROM number must be selected/switched on. If the watch is switched on, the FutureOS ROMs are not accessible. Therefore you need a OS function which first switch the watch on (like a ROM), then the watch data is read. At the end a FutureOS ROM is selected again. The same is true for the RTC of the M4 card, which can only be read by banking in the M4 ROM. The M4 RTC has a higher priority than the SmartWatch. LUHR does all the job.

**Attention:** Before you can use LUHR you have to copy it to RAM. This is done by the OS function DOBIN (see ROM A). If FutureOS is launched DOBIN is automatically called. But if you use the RAM of LUHR for your own programs, you have to call DOBIN to reactivate LUHR.

You want the latest version of this file?

Look the FutureOS homepage:

<http://www.FutureOS.de>