File: API-X-EN.DOC (7 pages) February 25, the CPC year 2019

Documentation of the Application Programming Interface of FutureOS

This file is a documentation of the part of the FutureOS API which can be permanently used in ROM A and / or all XROMs. This part of the API provides access to OS functions in FutureOS ROMs A-D for all expansion ROMs (XROMs) made for FutureOS.
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 any (Z80-) source code. Please refer also to the most recent LABEL - library file #EQU-API.ENG for the addresses of API entry points and 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 shall remain as they are.

3. ROM-number: provides the logical number(s) of the OS ROM(s) in which the corresponding OS function is located. More then one number is possible if dealing with API entrys present in ROM A and XROMs. This logical OS ROM number (A-D) may not be the physical ROM number!

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 can 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 routine.

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 located in ROM A and all connected XROMs.

The automatic start of Applications

Before we take a look at the OS functions of the XROMs we will show how to 'initialize' XROMs, which basically means to start applications automatically.
If an XROM is configured in the right way then every part of it, means every application (app.) in it can be started when the FutureOS itself gets started. This way you can for example load a wallpaper.
Beginning at label INITXROM newer XROMs contain an OS function, being called whenever FutureOS gets started and in case the XROM starts with the bytes &02, &09 at address &C000, marking it as FutureOS XROM.
This function will read the four bytes being located at address &FEFC in the XROM (see XROM Source-Code at Label KBAA). The first two bytes contain the control bits for automated start of applicatioins.
Beginning at KBAA+&02 the two bytes contain the autostart bits themselves. This is for 16 applications max. In both cased the least significant byte comes first in memory (usual Z80 addressing).
After the OS function did read the four bytes it checks if the controll bits are set correctly, if not it will exit.
For every app. start bit the corresponding control bit needs to be reset and this way confirmed.
In case the app. bit is set the Z80 register IY will be loaded with the value &9999 as mark. This way every application can check if this is the start of the whole OS. In this case the app. should return to the OS using the RET command after its execution.
In case of regular execution the app. should return to the OS by using a jump to the label TUR_E or KLICK (this is during regular work).
After loading IY with &9999 the app. will be executed. This way different tasks can be achieved when starting the OS, f.e. installing an wallpaper, screensaver or what ever you like.

<u>The construction of the autostart bits is shown here:</u>

Configuration bits (16) to Autostart Applications

```
KBAA  DW  %1111111111111111 ;Control bits, will be XOR connected
           0FEDCBA987654321 ;Number of the Application, LSB = 1. App.
      DW  %0000000000000000 ;Autostart bits, 0 = inaktive, 1 = Autostart on
```

If you want to automatically start the first application, then the bits will look like this:

```
KBAA  DW  %1111111111111110 ;Control bits, will be XOR connected
           0FEDCBA987654321 ;Number of the Application, LSB = 1. App.
      DW  %0000000000000001 ;Autostart bits, 0 = inaktive, 1 = Autostart on
```

Bytes beginning at Address: &FEFC: &FE, &FF, &01, &00

Further examples:

```
- Autostart of  2. Application, at &FEFC: &FD, &FF, &02, &00
- Autostart of  3. Application, at &FEFC: &FB, &FF, &04, &00
- Autostart of  4. Application, at &FEFC: &F7, &FF, &08, &00
- Autostart of  5. Application, at &FEFC: &EF, &FF, &10, &00
...
- Autostart of  8. Application, at &FEFC: &7F, &FF, &80, &00
- Autostart of  9. Application, at &FEFC: &FF, &FE, &00, &01
- Autostart of 10. Application, at &FEFC: &FF, &FD, &00, &02
...
- Autostart of 16. Application, at &FEFC: &FF, &7F, &00, &80
```

API: CALL FROM AN XROM TO AN OS FUNCTION IN FutureOS ROM A-D

Short description: A program located in an XROM can use this set of OS functions to CALL a regular OS function located in one of the FutureOS ROMs A-D.

Labels: ROM_X2A, ROM_X2B, ROM_X2C and ROM_X2D

ROM-numbers: ROM_A and all XROMs

Start addresses: ROM_X2A (&FD80), ROM_X2B (&FD99), ROM_X2C (&FDB2) and ROM_X2D (&FDCB)

Jump in conditions: IX = Target address of the OS function in ROM A-D
Anything else depends on the CALLed OS function.
The target ROM A-D is selected by using the appropriate label, which can be ROM_X2A, ROM_X2B, ROM_X2C or ROM_X2D.

Jump out conditions: This depends on the OS function which was called.

Manipulated: Flags and the RAM variable AKT_ROM.
Further Z80 registers can be altered, depending on the called OS function (see there).

Description: These four OS functions are part of the API which allows an FutureOS expansion ROM (XROM) to call OS functions located in the four core FutureOS ROMs A to D. First the XROM calls one of these four API entry points in ROM A. Then from ROM A they jump to ROMs A-D.
The entry points ROM_X2A, ROM_X2B, ROM_X2C and ROM_X2D are located only in ROM A (not in B, C or D). And of course they must be part of all XROMs (see example Source Code).
The selection of the target ROM (A-D) which contains the OS function you wish to call is managed by using the appropriate label: ROM_X2A for ROM A, ROM_X2B for ROM B, ROM_X2C for ROM C or ROM_X2D for ROM D.
The jump in and jump out conditions as well as which Z80 registers are manipulated depend solely on the target OS function you want to call.
Careful: Since the target address of the OS function has to be given in register IX you can not use OS functions which need te receive parameters in IX.
After the return the Z80 jumps back to the XROM which was originally calling the OS function.

Attention: All four of these OS functions must be present in any XROM you are creating. Please refer to the provided example source code.
When you assemble an XROM you need to provide the correct hardware ROM select numbers for all FutureOS ROMs. Or you can adapt them later on.

The OS function you wish to call in the core OS (A-D) is not allowed to alter the RAM variable AKT_ROM. Else this OS functions would jump back to a wrong XROM which would result in a crash. But most of the OS functions do not alter AKT_ROM. :-) **Example:**

```
LD   IX,HOLE0ID  ;Load address of the OS function in register IX
CALL ROM_X2B     ;This OS function is located in ROM B. Call it there!
```

Show TEXT from an XROM on the SCREEN - CONTROL CODES are REGARDED

Short description: This set of OS functions can be used to display text which is located inside an XROM on the screen. You can use screen MODE 1 or 2 with different colors / attributes. Here control codes are regarded and will exert their function.

Labels: TXR_2, TXR_2D, TXR_2I, TXR_2K, TXR_2U, TXR_GB, TXR_GG, TXR_BB and TXR_RR.

ROM-numbers: All XROMs

Start addresses: TXR_2 (&FDE4), TXR_2D (&FDF5), TXR_2I (&FE07), TXR_2K (&FE19), TXR_2U (&FE2B), TXR_GB (&FE3D), TXR_GG (&FE4F), TXR_BB (&FE61) and TXR_RR (&FE73)

Jump in conditions: HL = Addresse of the text which shall be displayed on the screen (like for regular OS functions which display text and use control codes).
However, the first byte of the text contains the length of the text from 1 to 255.

Jump out conditions: The text was displayed on the screen.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL' and the RAM variables C_POS and AKT_ROM.

Description: The text output functions of the OS are located in ROM A.
Therefore its not possible to display text being located in an XROM directly.
This set of OS functions will first copy the text from the XROM to the text buffer (&B000-&B7FF) and subsequently display it on screen. As usual the string must be terminated with the byte &00 (see example).
However keep in mind that the first byte of the text string needs to contain the length of the text (1-255 characters).Like usually the start of the text need to be given in register HL. Here an example:

```
LD   HL,TEXT     ;Pointer to text string
CALL TXR_2       ;Output text in MODE 2 normally

DB 18,"This is one test!",&00 ;Length of text, text string itself
```

You can use the following OS functions for text output:

- **&FDE4 TXR_2 - MODE 2 - normal**
- **&FDF5 TXR_2D - MODE 2 - streaked out**
- **&FE07 TXR_2I - MODE 2 - inverted**
- **&FE19 TXR_2K - MODE 2 - italics**
- **&FE2B TXR_2U - MODE 2 - underlined**
- **&FE3D TXR_GB - MODE 1 - Pen 1 / Pen 2**
- **&FE4F TXR_GG - MODE 1 - Pen 1**
- **&FE61 TXR_BB - MODE 1 - Pen 2**
- **&FE73 TXR_RR - MODE 1 - Pen 3**

The screen mode and text attributes are selected by using the corresponding OS function.

Attention: The first byte must contain the length of the text. Control codes are regarded. Strings must be terminated with the byte &00.

Show TEXT from an XROM on the SCREEN - CONTROL CODES are IGNORED

Short description: This set of OS functions can be used to display text which is located inside an XROM on the screen. You can use screen MODE 1 or 2 with different colors / attributes. Here control codes are ignored and displayed as character on screen.

Labels: SXR_2, SXR_2I, SXR_2U, SXR_GB, SXR_GG, SXR_BB and SXR_RR

ROM-number(s): All XROMs

Start addresses: SXR_2 (&FE85), SXR_2I (&FE96), SXR_2U (&FEA7), SXR_GB (&FEB8), SXR_GG (&FEC9), SXR_BB (&FEDA) and SXR_RR (&FEEB)

Jump in conditions: BC = Length of strings + 1 (maximum 2000)
HL = Address of the text (like using a regular text output function)

Jump out conditions: The text was displayed on the screen.

Manipulated: AF, BC, DE, HL, AF', BC', DE', HL' and the RAM variables C_POS and AKT_ROM.

Description: The text output functions of the OS are located in ROM A. Therefore its not possible to display text being located in an XROM directly.
This set of OS functions will first copy the text from the XROM to the text buffer (&B000-&B7FF) and subsequently display it on screen.
Here control codes are ignored and displayed as characters on screen.
In this case the Z80 register BC contains the lengt of the text +1.
And like usual the start address of the text needs to be given in register HL.

Here an example:

```
LD   HL,TEXT     ;Pointer to text
LD   BC,27+1     ;BC gets the length of the text +1
CALL SXR_2       ;Output text in MODE 2 normally on screen

DB "This is just one more test!" ;This is the text string
```

The following OS functions can be used:

```
&FE85 SXR_2  - MODE 2 - normal
&FE96 SXR_2I - MODE 2 - inverted
&FEA7 SXR_2U - MODE 2 - underlined
&FEB8 SXR_GB - MODE 1 - Pen 1 / Pen 2
&FEC9 SXR_GG - MODE 1 - Pen 1
&FEDA SXR_BB - MODE 1 - Pen 2
&FEEB SXR_RR - MODE 1 - Pen 3
```

The screen mode and text attributes are selected by using the corresponding OS function.

Attention: Control codes are ignored, they will be shown as characters on screen.

You want the latest version of this file?

Look the FutureOS homepage:

http://www.FutureOS.de