knobloch electronic

fischertechnik Robo Interface

&

"C"

Instructions for Use of the Renesas C Compiler for the Robo Interface

Version 1.66a
Status: May 15, 2006

Version Interface Firmware: 1.66.0.3
Version Renesas High Performance Embedded Workshop: 4.00.03.001

Knobloch GmbH
Weedgasse 14
55234 Erbes-Büdesheim

entwicklung@knobloch-gmbh.de
www.knobloch-gmbh.de

1 Table of Contents

2 Preface

The Robo Interface has a 16 bit microprocessor from the M16C series from the manufacturer, Renesas, type M30245, This provides a complete development environment under the name "High Performance Embedded Workshop." Since the development of these professional tools is very expenditure intensive, but, on the other hand, they are only used by a "small" development group, the purchase price of 2000 euros and more for such a version is certainly normal.

Renesas has introduced a new license model with the current version 4. You can use the developer package without limitations for a period of 60 days after the first installation. After this, the compiler only creates programs with a maximum size of 64 Kbyte. This code size should be more than sufficient for most of the fischertechnik projects. This allows you to use a professional tool with a lot of possibilities at no cost.

Due to the size of the installation file, which is about 80 Mbyte, it is not contained in this package. You can download the software in the Internet directly from Renesas. If you do not have suitable Internet access, you can order a CD with the installation file from the fischertechnik Individual Parts Service at (vertrieb@knobloch-gmbh.de or www.knobloch-gmbh.de. You can obtain the CD at a price of 1.00 euros plus the normal service charge for processing and shipping. Item number for the CD is 79028 status as of January 2006.

In addition to the installation file for the development environment, the data package FtCComp.ZIP is required. In addition to these instructions, this data package also contains example programs and a program (FtLoader) to allow you to store the program file, which is created by the C Compiler, in the interface. This file is also contained on the CD. In addition, the FtLib is also on the CD.

The example programs are also intended to serve as the basis for your own projects. With this information, the programer with C basic knowledge can create C programs and store these in the interface. However, there are some dangers here, which we will discuss in the chapter, "Low Level Programing."

3 Robo Interface & "C"

Up to three programs can be stored in the Robo Interface per download. Program 1 and program 2 are permanently stored in a flash memory and a third program can be stored in the RAM. The RAM is erased when a flash program is started and if a power outage occurs on the interface.

The selection of the active program is done with the program button. If this button is pressed for longer than 0.5 seconds, the desired program can be selected. The program LED for programs 1 and 2 light up sequentially. If a program is stored in the RAM, this is indicated by the illumination of both LEDs. If the memory location is empty, then the corresponding program cannot be run.
To start or stop the program, which is displayed, you must press the program button for a short time (< 500ms). Programs can also be started or stopped through the PC interface with the program, FtLoader.

4 PC Interfaces

4.1 General

The selection of the interfaces is done by pressing a button on the Robo Interface. After it is turned on the "AutoScan" mode is active. The USB, serial interface and the radio module (if installed) are checked to see if the data are available. This condition is indicated by the illumination of the interface LEDs.

When an interface transmits data then the other interfaces are blocked. The active interface blinks to indicate the transfer of data. If no data flow over the active interface for longer than 300 ms, the AutoScan mode is switched on again.

If you press the "Port" button, then the next mode is selected according to the following table.

1.	AutoScan	USB serial radio[1]
2.	AutoScan	USB serial
3.	USB	
4.	Serial	
5.	IR direct	

1
This mode is only activated if the radio module is installed.

If the port button is pressed for more than three seconds, the interface switches to the "intellgent interface online mode." The serial interface then works with the parameters 9600,n,8,1. To show the mode, the "SER" LED blinks very fast. In this mode, the interface acts like an intelligent interface in the online mode. However, no programs can be downloaded. If you press on the "Port" button, the AutoScan mode is set again.

In the passive movde, the active Robo Interface is controlled through a serial interface (RS232), the USB or radio.

4.2 Several USB Intefaces on one Computer

In order to operate several interfaces on the USB, first each interface must be assigned its own serial number. As a standard, all interfaces are delivered with the same serial number in order to avoid problems with the exchange of interfaces. The Windows operating system, however, recognizes only those interfaces with different serial numbers. For "each" serial number, the corresponding driver is installed. To do this with Windows 2000 or XP, administrator rights are necessary.

Therefore, as a standard, all ROBO Interfaces and ROBO I/O Extensions are delivered with the same serial number. As long as only one interface is used on a computer, there are no problems, if the interfaces are operated on different computers. The computer differentiates the products by their names such as ROBO Interface, ROBO I/O Extension and Robo RF DataLink and their particular serial number. Therefore, a Robo interface and a Robo I/O extension can be operated at the same time on a computer without changing the serial number because these are different products.

However, if several of the same products such as Robo Interfaces are to be operated on one computer through the USB, then before this is done the serial numbers of the interfaces must be changed so that these are seen as different by the computer.

Note! If connected to the serial interface on the computer, then no serial numbers must be changed.

The interface has therefore stored two serial numbers. Using the software, the setting can be made to have the standard serial number "1" be active or the device serial number "2," which the manufacturer programs, to be active when the device is turned on.

The changing of the serial number can be done using the software FtDiag.exe, RoboPro or the function, GetFtDeviceSetting() or SetFtDeviceSetting() of the FtLib.

To change the serial number, only one product may be connected to the USB because otherwise the computer cannot recognize the difference between them. In FtDiag.exe, call up "SCAN USB" and then click on the "USB Device" button and then go to the menu Properties/Setup. In this window, you can set the desired serial number, which will be active after you turn it on the next time.

Caution: If the serial number is changed, then the next time the interface is turned on, the Windows driver may have to be reinstalled. To do this with Windows 2000 or XP administrator rights are necessary. If you don't have these, you can't install the driver and thus you would not be able to access the interface through the USB. In this case, you can press and hold the PROG button when you turn on the interface. The interface then uses the serial number "1" and is then recognized by the installed driver, for example, to change the serial number to permissible values.

Note! The serial numbers, which are printed on the products, are hex numbers.

Even, if the USB "theoretically" allows up to 127 devices, practice has shown that with Windows XP with SP1 on a three GHz Pentium 4 only about four to five products can be reliably operated parallel, which means that updating time for the transfer area is a maximum of 10 ms. With Windows 98, this is up to 10 devices. This is due to the fact that the internal Windows drivers for the motherboard hardware under XP are not optimized for "real time applications." Therefore, it is more useful to connect IO extensions to the Robo Interfaces instead of each product individually to the USB.

5 Installation of the Development Environment

The installation environment can be downloaded at the following link:

http://eu.renesas.com/fmwk.jsp?cnt=quicklink_updates_product_child.htm&fp=/products/tools/coding_tools/c_compilers_assemblers/m3t_nc30wa/child_folder/&title=QuickLink%20Updates

You can also find this link in the file ""Link.txt" of the FtCComp.zip and this will save you the tiresome copying.

Almost 80 MB of the "hew4_nc30" packet must be downloaded.
This is the "Renesas C Compiler Package for M16C family V.5.4.00 Release 00 with High Performance Embedded Workshop (HEW) V.4".

Note!
It is possible that Renesas may change this link if a new version is made available. Then you must look for the development environment for M16C/24 (M16C/20 series) on the Renesas Website. Up to now, you could download the C Compiler without registering. However, since you can only download from the Japanese site if you register, it is possible that "Renesas Europe" will likely require this as well.

You can also find alternative addresses (where available) in the file "Link.txt."

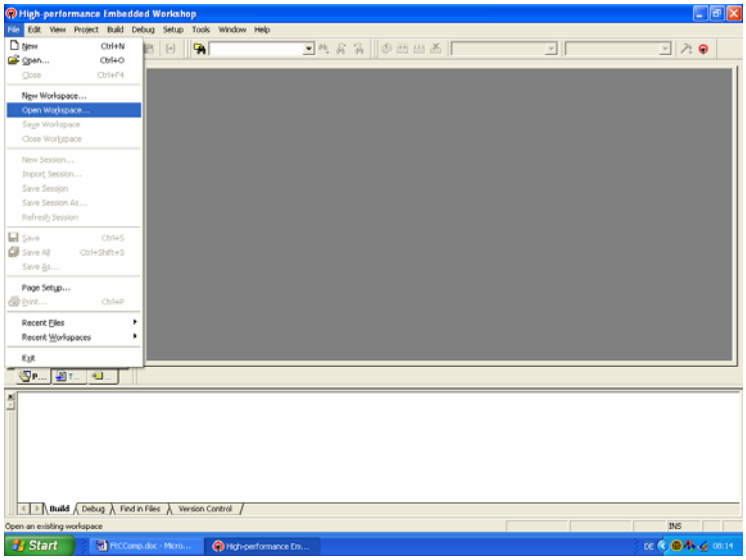
The precise installation of the package is described in separate instructions in the file "Renesas-Install.pdf."

6 Five Steps to the First C Program

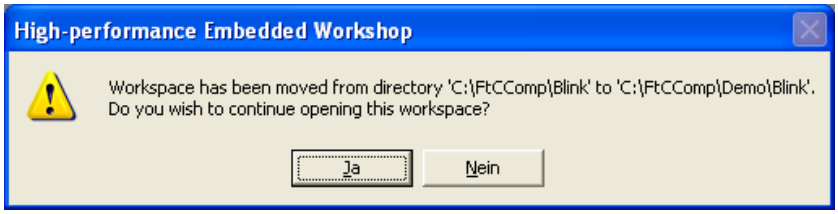
After the Renesas development environment has been installed, the FtCComp example programs can be copied to the harddrive in a separate folder. In these instructions, we use the folder c:\FtCComp for the following examples. We have the contents of the FtCComp.zip file unzipped there.

The following five steps show the path to the first C program in the Robo Interface.

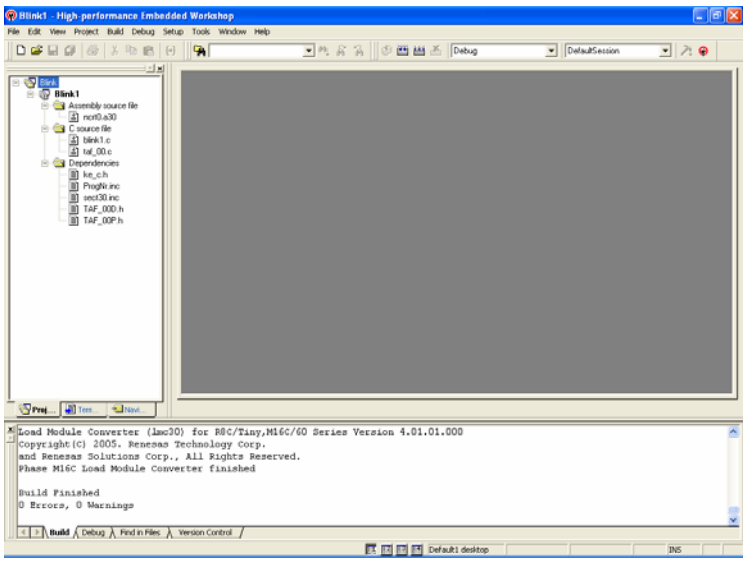
Important Information
The Robo interface must have the firmware 01.58.00.03 or higher because otherwise there is no support for own programs in the firmware.



1. Start the Renesas development environment, "High Performance Embedded Workshop," and through "File / Open Workshop" load the file "Blink.hws" in the folder "C:\FtCComp\Demo\Blink."

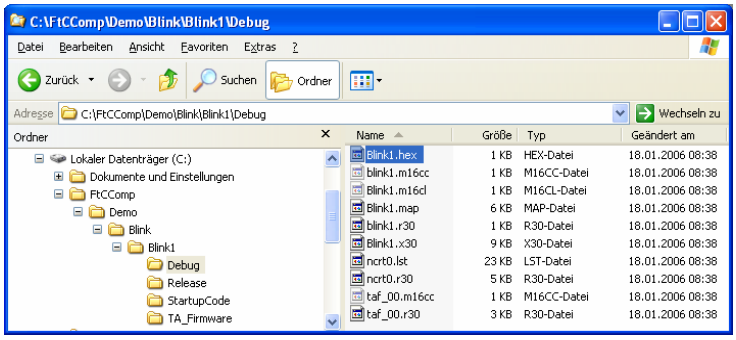


   If you have installed the example files in another folder, you will receive a message that the file was stored in a different folder and has now been moved. You can continue by pressing "Yes."

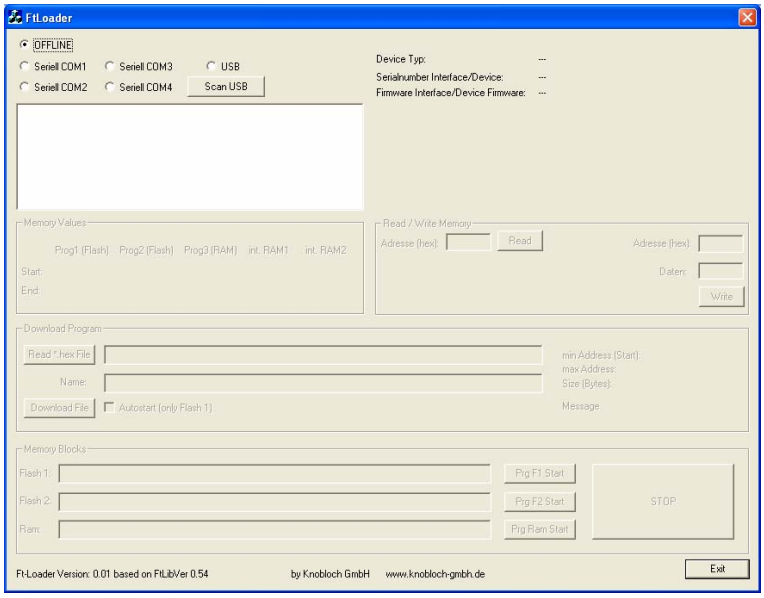


2. Now compile the example program by clicking on the button, "Build All," (left next to "Debug") or use "Build / Build All" in the menu bar (key "F7" also works).

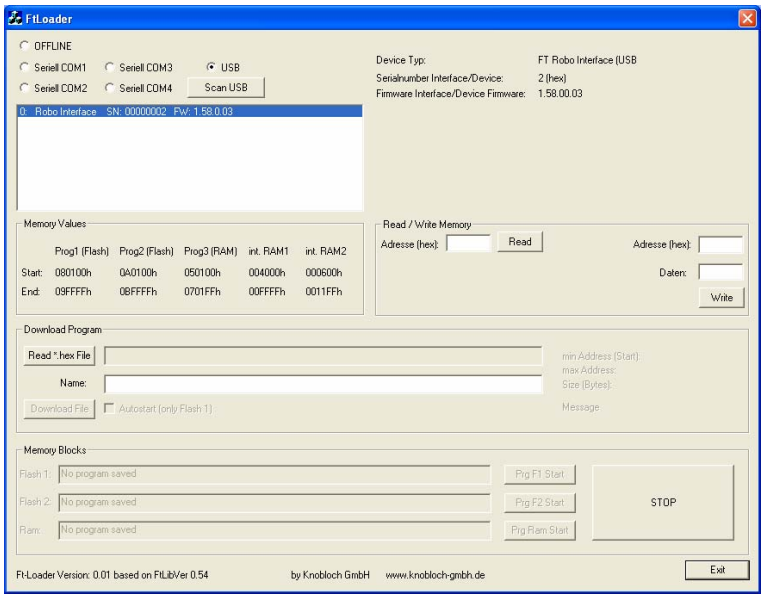
   In the lower output window, some messages are now running through very quickly. When the compiler and the linker are finished, "0 Errors" should appear there. As soon as the trial period of 60 days has expired, a "warning" always appears that tells you that only programs with a size of 64 k can be generated.



- The actual program for the Robo Interface is stored in the file "Blink1.hex" in the folder "C:\FtCComp\Demo\Blink\Blink1\Debug."

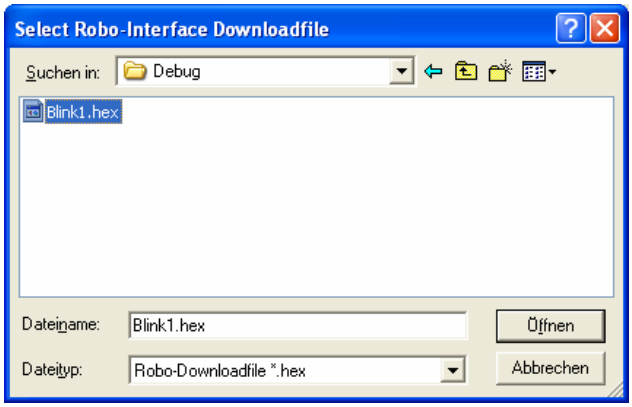


3. Start the program, FtLoader.exe, which is located in the folder, FtCComp. The program file is stored in the interface with this program.
   After the interface is connected to the computer and is supplied with power, the program can be stored in the interface.

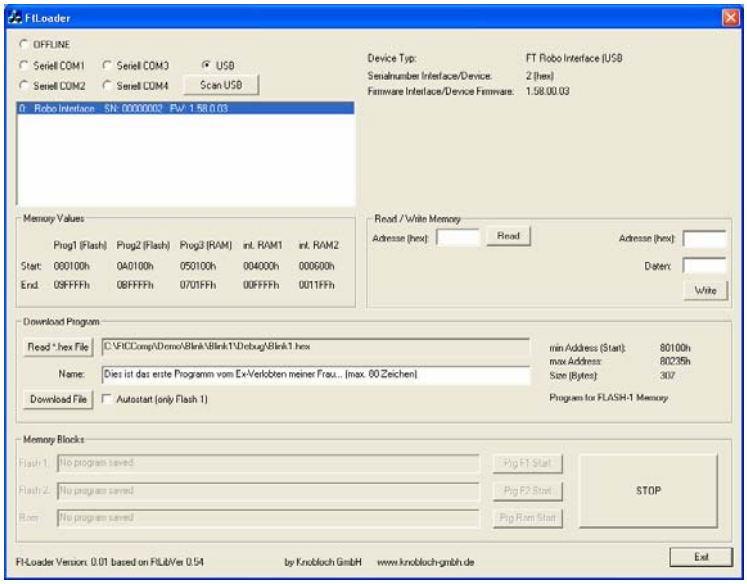


In this example, the interface is connected to a USB and if you click on the "Scan USB" button, a search is made for all interfaces. Following this, mark the desired interface so that it is highlighted.

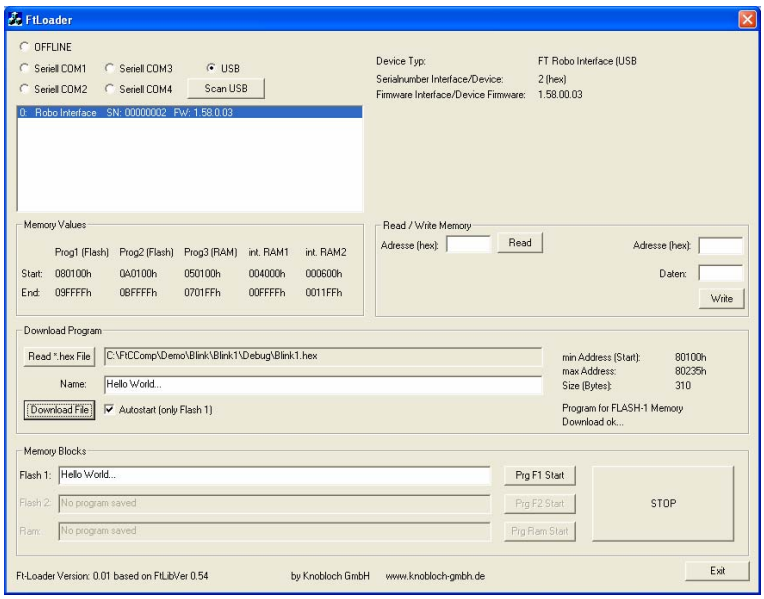
The program now queries the current memory areas of the interface and shows the values as information in the "Memory Values" window. In the lower window, "Memory Blocks," the status of the three memory areas of the interface is shown. A name with a maximum of 80 characters can be stored for each program.



4. Now open the file, "Blink1.hex," in the folder "C:\FtCComp\Demo\Blink\Blink1\Debug" by pressing the "Read *.hex File" button.



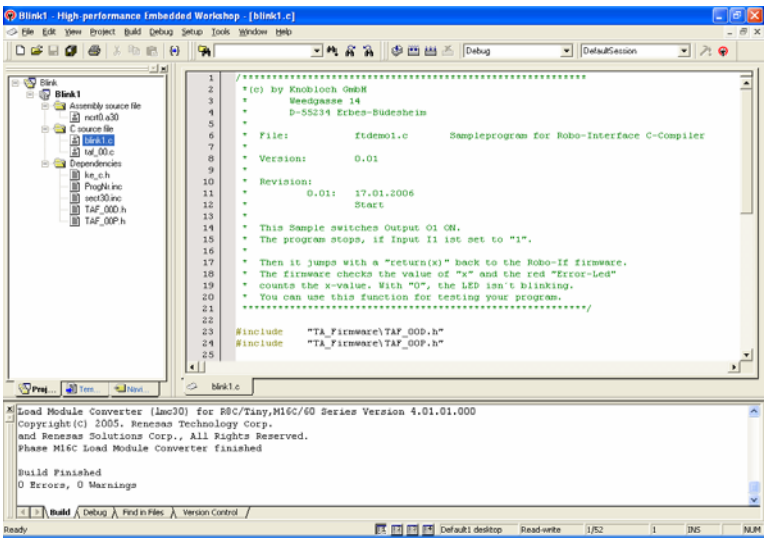
- The file is read in and using the addressses it is automatically determined, for which memory area the program is intended. In this example for the area "Flash 1". A name with up to 80 characters can be stored as a descriptive text.



5. The download button stores the program in the interface together with the name and the display is also updated. The program is now stored in the memory location "Flash 1." It can be started with the "Prg F1 Start" button and ended with "Stop."

   Congratulations!
   You have now created "your" first program and stored it in the interface.

   On the next page, we will make a closer examination of the way that the "Blink1" program works.



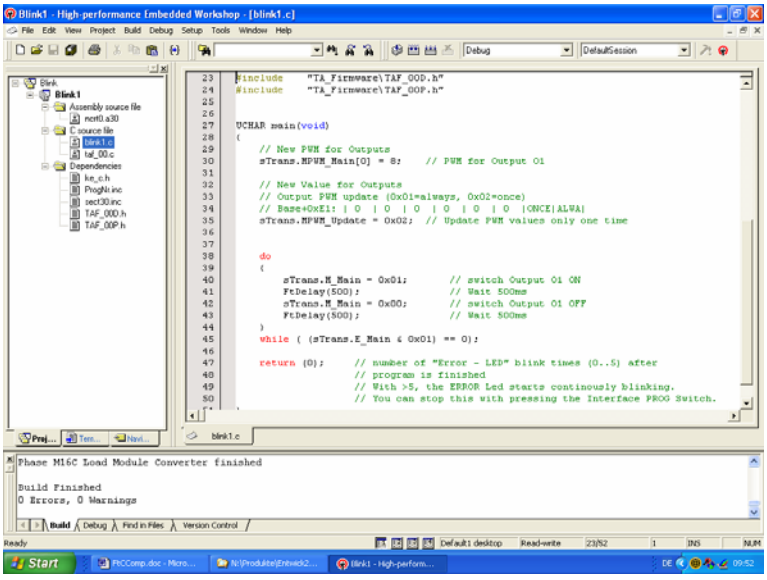
- By clicking on the file name "Blink1.c," we open the program file. This is shown in the window on the right.
  Two important lines follow the description, which is displayed in green:

```
#include     "TA_Firmware\TAF_00D.h"
#include     "TA_Firmware\TAF_00P.h"
```

  In the header file "TAF_00D.h," there is a definition of the "transfer area," a data structure, through which the program "communicates" with the hardware in the interface.

  In the header file "TAF_00P.h," there is a definition of the firmware macros to call up functions within the firmware.

  Both of these files must therefore be contained in every program. In addition, the file "TAF_00.c" must be added to every program. You can find this in the area on the left under "C source files."



```
UCHAR main(void)
{
    // New PWM for Outputs
    sTrans.MPWM_Main[0] = 8;    // PWM for Output O1

    // New Value for Outputs
    // Output PWM update (0x01=always, 0x02=once)
    // Base+0xE1: | 0  | 0  | 0  | 0  | 0  | 0  |ONCE|ALWA|
    sTrans.MPWM_Update = 0x01;  // Update PWM values every 10ms


    do
    {
        sTrans.M_Main = 0x01;       // switch Output O1 ON
        FtDelay(500);               // Wait 500ms
        sTrans.M_Main = 0x00;       // switch Output O1 OFF
        FtDelay(500);               // Wait 500ms
    }
    while ( (sTrans.E_Main & 0x01) == 0);

    return (0);     // number of "Error - LED" blink times (0..5) after
                    // program is finished
                    // With >5, the ERROR Led starts continously blinking.
                    // You can stop this with pressing the Interface PROG Switch.
}
```

- The "actual" program "main" starts after both header files.

  ```
  sTrans.MPWM_Main[0] = 8;    // PWM for Output O1
  ```

  First, the speed value "8" is assigned to the output 1.

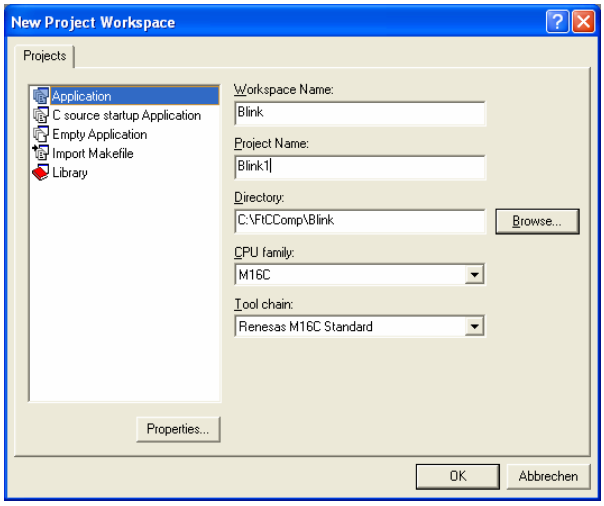
  ```
  sTrans.MPWM_Update = 0x02;  // Update PWM values only one time
  ```

  Since the calculation of the PWM values for the control of the outputs takes some time, the firmware must be notified when a new calculation is to take place. This is now done in this line. With the value 0x01, the firmware recalculates the PWM values every 10 ms and for the value 0x02 only one time. The bit is erased after the calculation.

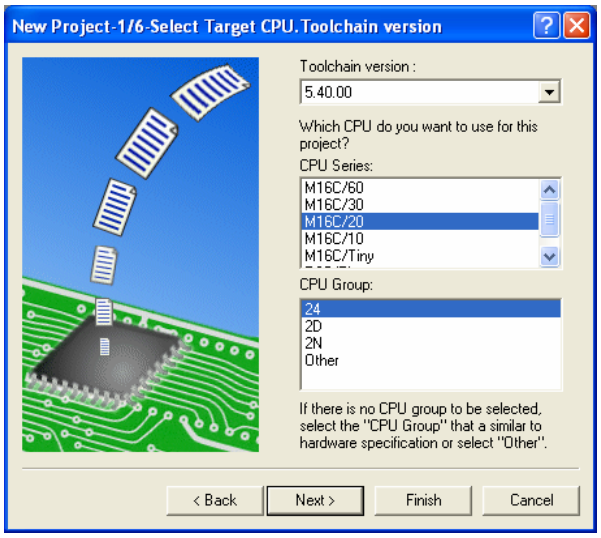
  In the following `do..while` loop, output 1 is turned on and off for 500 ms. As soon as the input I1 is set, the program is ended. With the return instruction, a numerical value can be transmitted to the firmware, which decides how often the red error LED is to blink. For "0" it remains off, 1 to 5 corresponds to the number of blinks and for a value of >5 the LED blinks continuously until the "PROG" button is pressed one time.

7 Generating a New Project

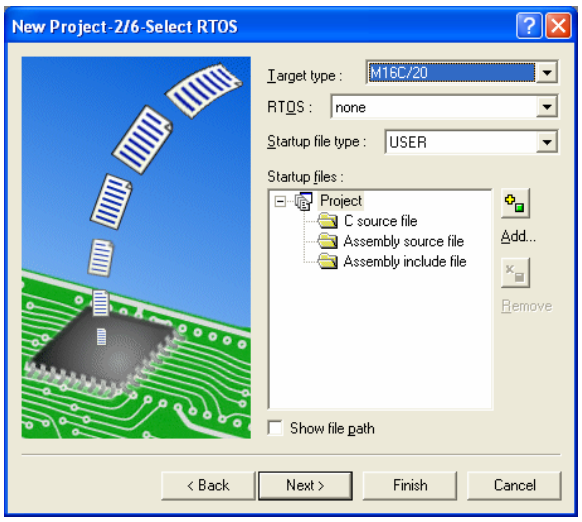
- If a project is opened, close the current workspace (File / Close Workspace) in the High Peformance Embedded Workshop (HEW).
- Create a new workspace (File / New Worspace)



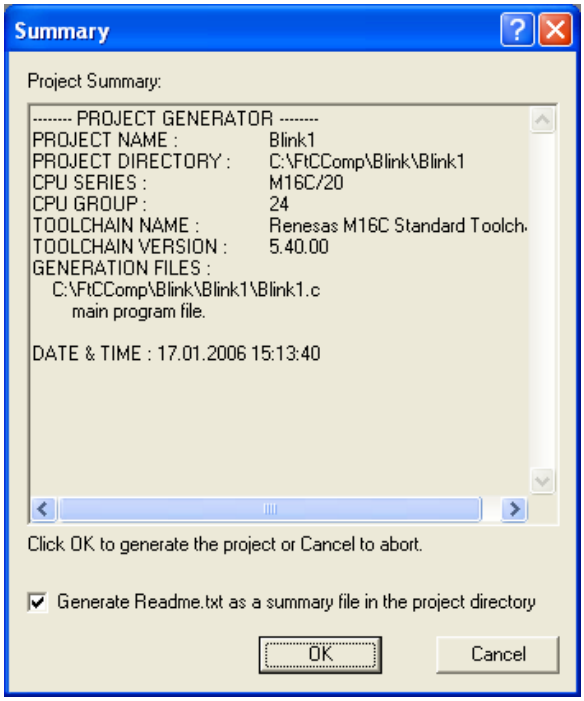
- First, you should select the folder,for example, c:\FtCComp.
- In this example, we give the workspace the name "Blink" and the "Blink" folder in the "FtCComp" directory is automatically recommended.
- We name the project "Blink1".
- The CPU family is "M16C" and the tool chain is "Renesas M16C Standard."



- At this point, the CPU series "M16C/20" with the CPU group "24" must be selected. Then press "Next."
  With "Finish" some settings are not queried, but we need these.

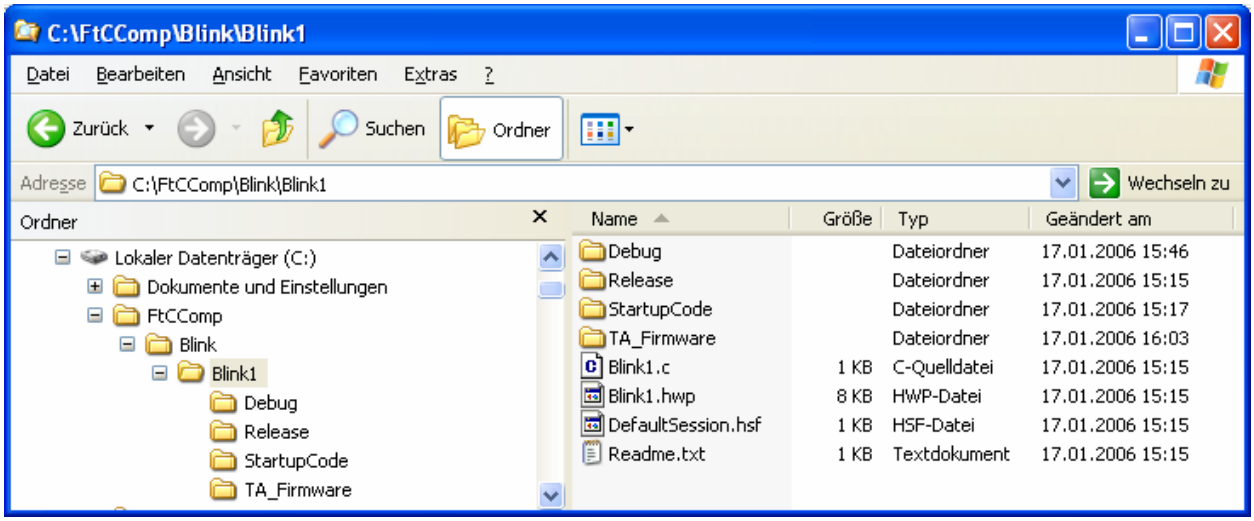


- The "Target type" must now be set to "M16C/20" and the "Startup file type" to "USER." Then select "Finish."

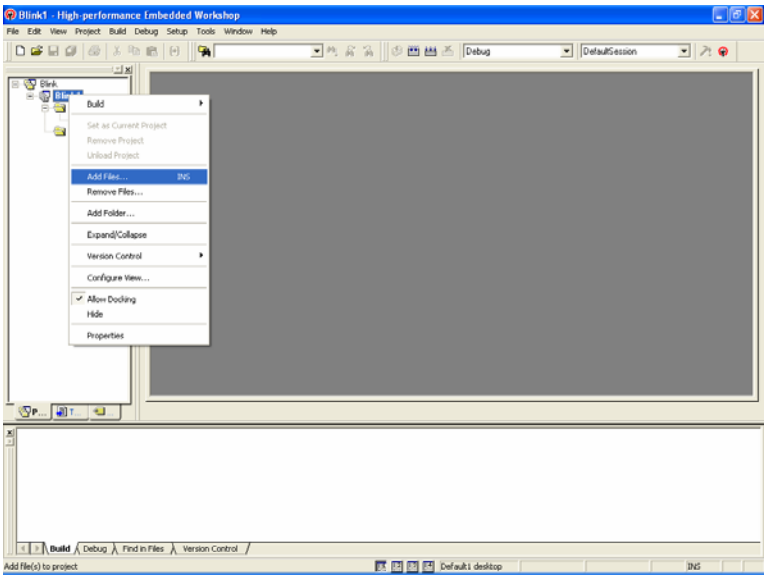


- The summary appears and with "OK" the project is generated.

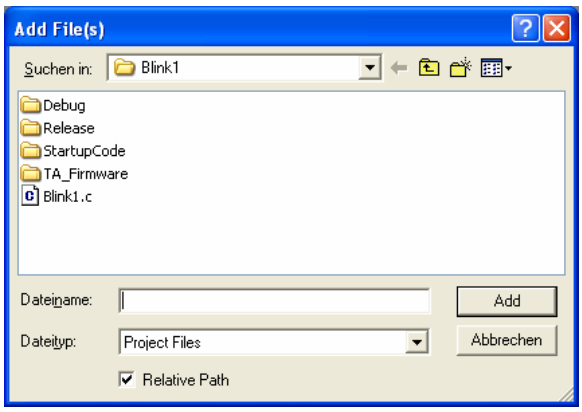
- The C Compiler needs some specific files for Robo Interface for the later C program and these files are located in the folders "StartupCode" and "TA_Firmware" for the examples.



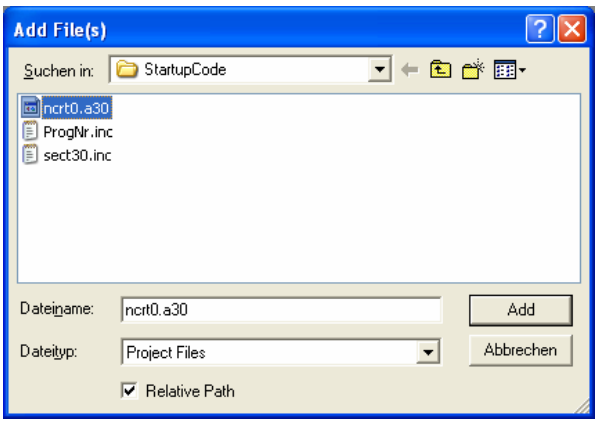
- Now, copy these folders with Windows Explorer to the "Blink" directory in C:\FtCComp\Blink\Blink1.



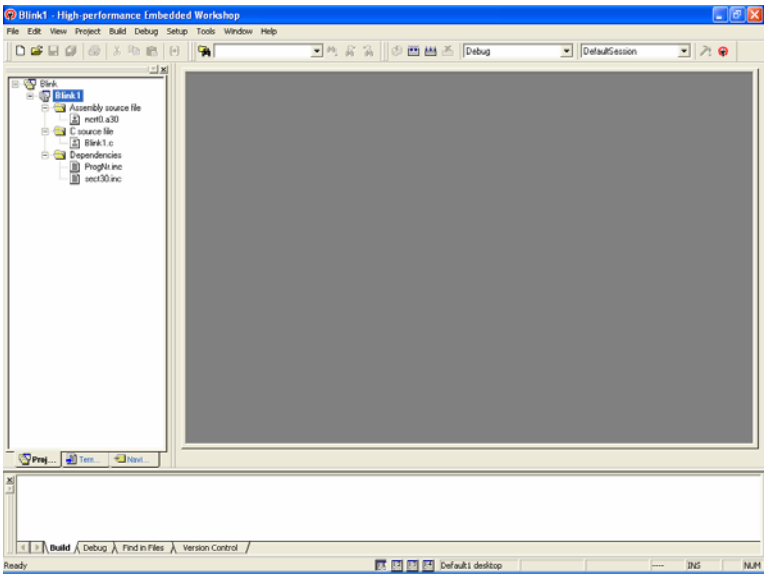
- The project must now be notified of these files. The start files can be added with a right mouse click on the project name "Blink1" and selecting "Add Files" from the menu.



- The "Add Files" menu opens. Here, change to the folder "StartupCode" and mark only the file "ncrt0.a30." The other two files "ProgNr.inc and sect30.inc" are not to be marked.

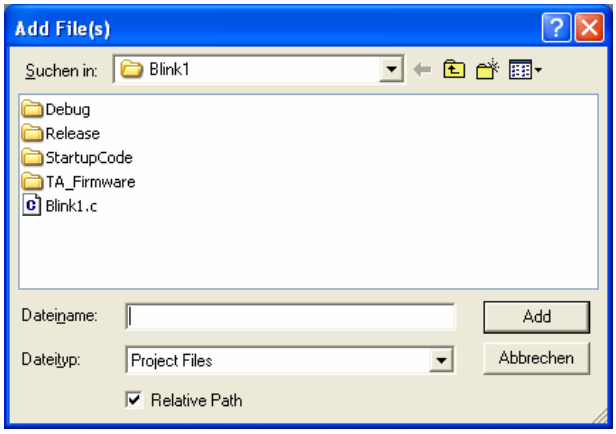


- Use the "Add" button to add this file to the project.

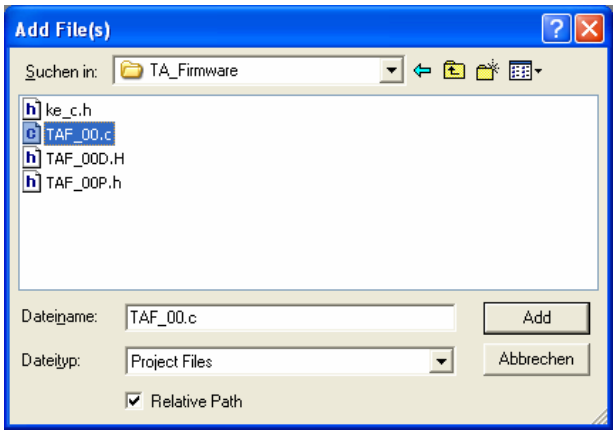


- Depending on the file type, the "Add" function adds the file to a corresponding folder in the project tree. The files are "analyzed" at the same time by the "Add" function. You can recognize this in that both include files "ProgNr.inc and sect30.inc" are in the "Dependencies" folder.
  Note! In the FtCComp examples, some folders were created and the files were correspondingly structured. In this example, we retain the automatic order.

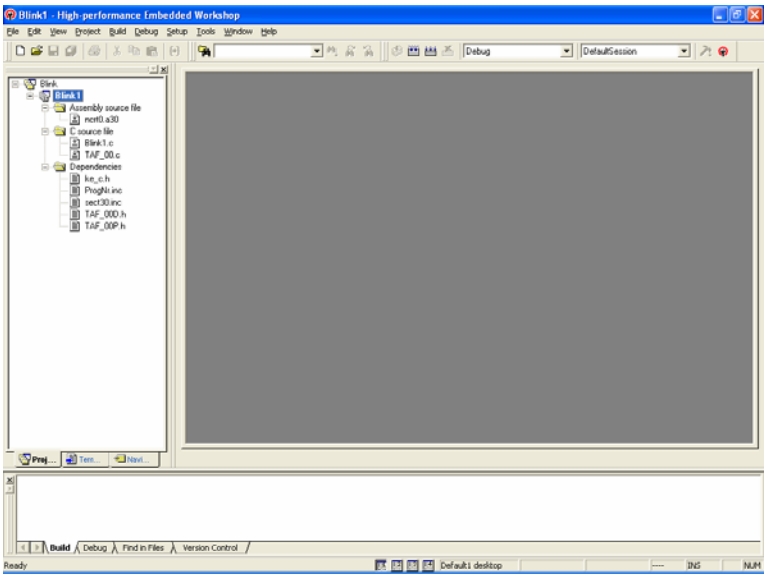
- In the next step, the files from the TA_Firmware folder are to be added. Here, you also click with the right mouse button on the project name "Blink1" and "Add Files" to add the TA_Firmware files.

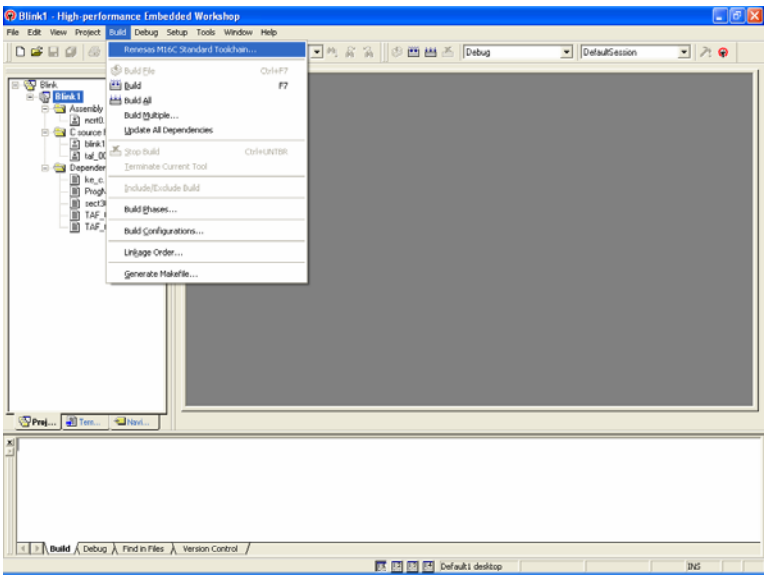


- The "Add Files" menu opens. Here, change to the folder "TA_Firmware" and mark only the file "TAF_00.c." The other files are not marked.

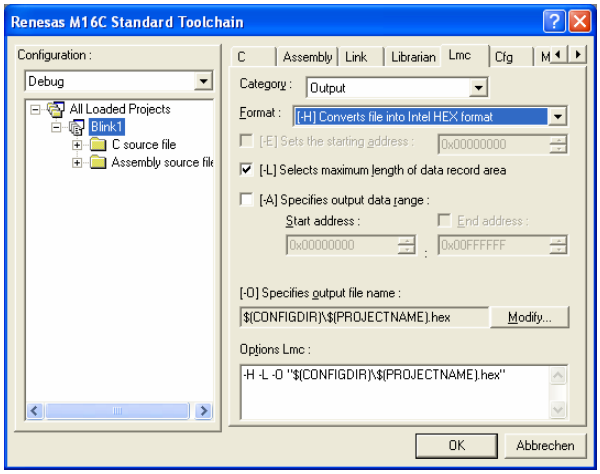


- Use the "Add" button to add this to the project.

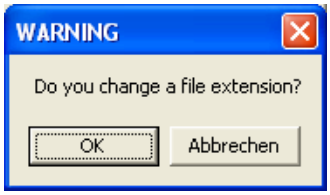




- Now the development environment must be notified that we require the output file in the "HEX" format. For this purpose, open "Build / Renesas M16C Standard Toolchain."

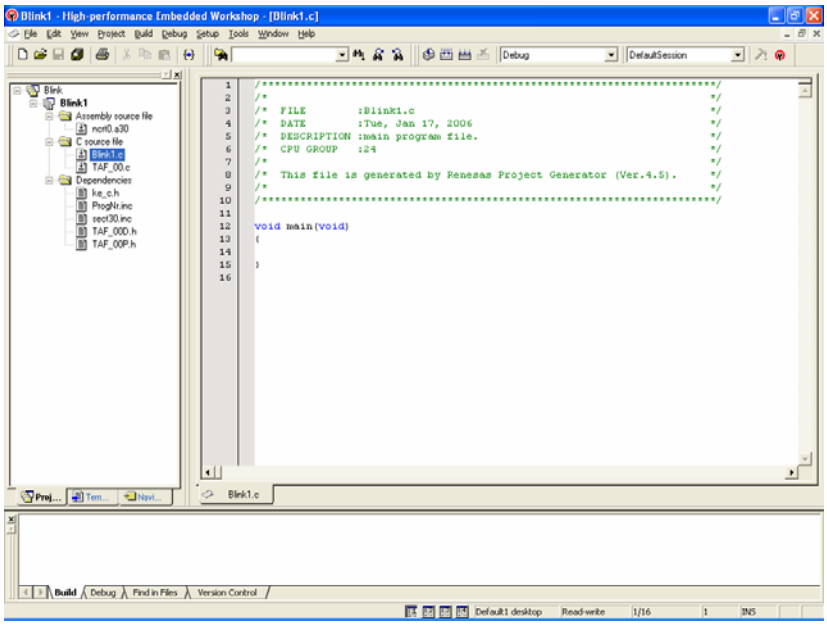


- In the tab "Lmc" in the category "Output," the "Converts file into Intel HEX format" must be selected.



- Confirm the "Warning" information with "OK."
- Then, confirm the setting on the "Lmc" tab with "OK."

- Now that this "preliminary work" has been completed, the "actual" programing can start. To do this, double click with the left mouse button on the "C source file" file "Blink1.c."



- The file opens in the window on the right. This was created automatically when the project was generated. So that the program can also control the hardware for the interface, two important lines must follow the description (shown in green):

```
#include    "TA_Firmware\TAF_00D.h"
#include    "TA_Firmware\TAF_00P.h"
```

  In the header file "TAF_00D.h," there is a definition of the "transfer area," a data structure, through which the program "communicates" with the hardware in the interface.

  In the header file "TAF_00P.h," there is a definition of the firmware macros to call up functions within the firmware.

  Both of these files must therefore be contained in every program. In addition, the file "TAF_00.c" must be added to every program. You can find this in the area on the left under "C source files." This can be added to the project using the "Add" function.

  The program interface is normally used for the development of "Embedded Systems." For example, control software is prepared for heating control in an apartment house. Normally, you cannot "end" such software and therefore a "main" function is started one time after it is turned on and should continue to work. Every C program starts with the "main()" procedure. Therefore, do not change this name.

  In the Robo interface, the program may end itself if this is necessary. When ending, the firmware expects a numerical value of 0 to 225. This value determines if the red error LED "blinks" after the program stops. Some changes must be made so that this works.

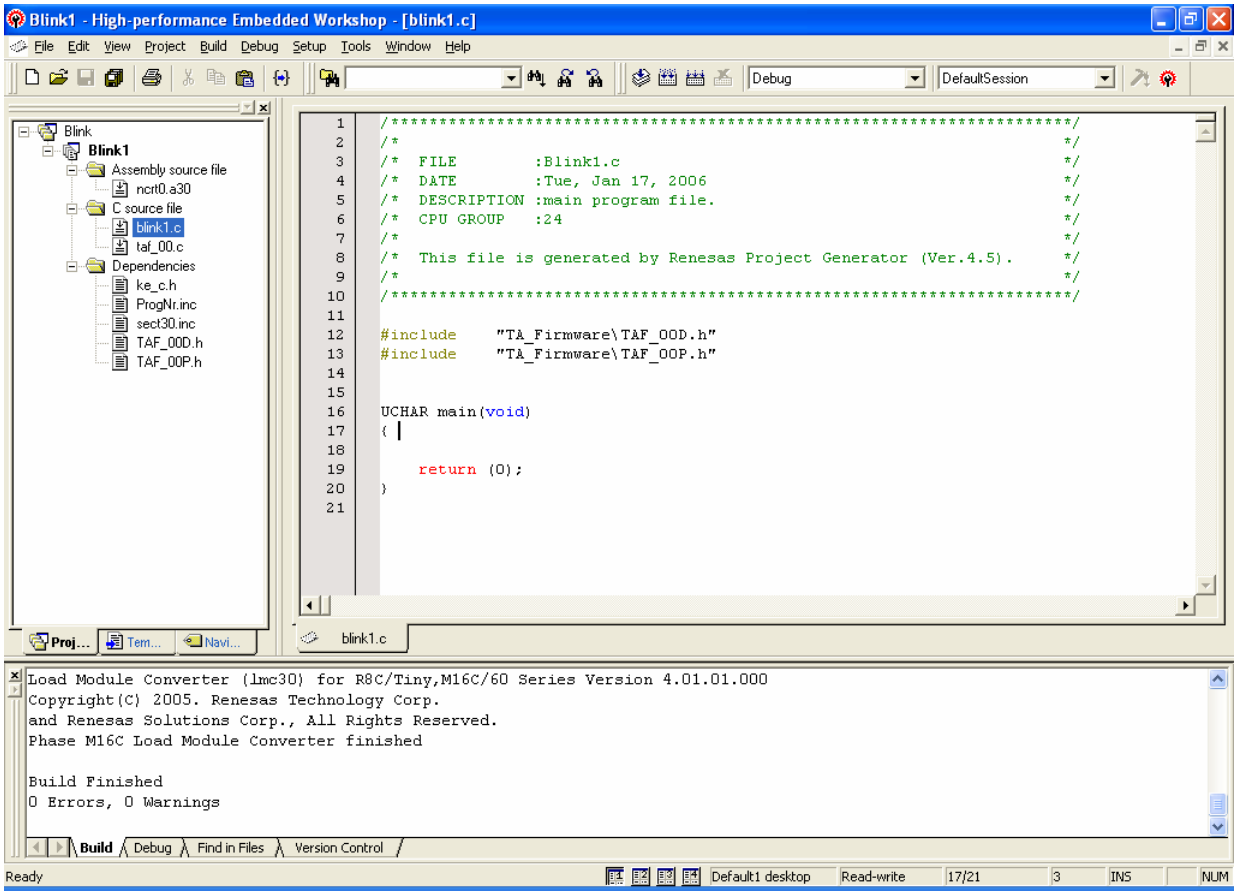
The "void" before ""main" must be changed to "UCHAR" so that the function can provide a value.

```
UCHAR main(void)
```

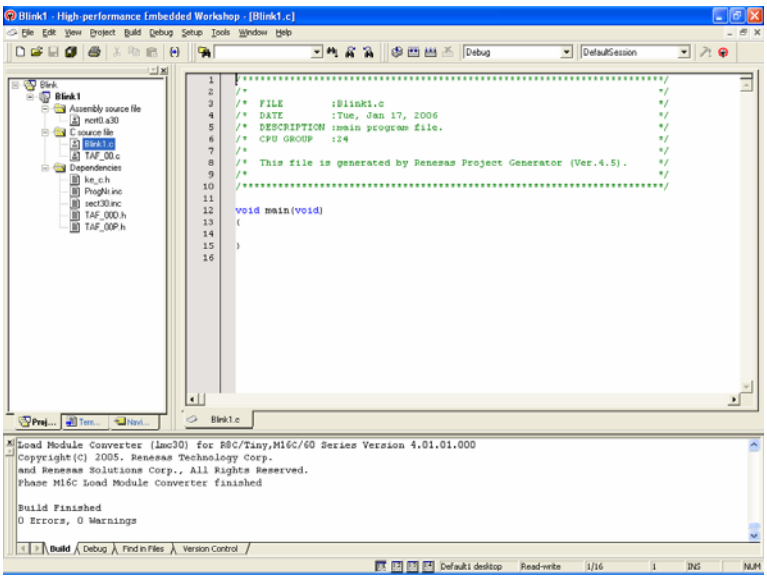
Then, at the end of the main fucntion a "return" with a value must be given.

```
return (0);
```

The result now looks like this.



- Now the first test: click on the button, "Build All," (left next to "Debug") or use "Build / Build All" in the menu bar (key "F7" also works). The compiler now translates the program into machine instructions.



- In the lower output window, some messages are now running through very quickly. When the compiler and the linker are finished, "0 Errors" should appear there. As soon as the trial period of 60 days has expired, a "warning" always appears that tells you that only programs with a size of 64 k can be generated.

- Now that it is certain that the instructions are correct, you can start with the preparation of the program. You can find the finished Blink1 program in the examples. Also please note the instructions in the following chapter.

8 Instructions for Programing

The querying of the inputs and control of the outputs is done through a "transfer area" (communication memory area). This area is matched with the hardware every 10 ms through the interface firmware after the program is started. Programers who have already worked with the FtLib are familiar with this principle. The structure of the transfer area is found in the folder, TA_Firmware, in the header file "TAF_00D.h." The transfer area is identical to the FtLib version. The start address for the transfer area is fixed in the firmware at 0x400 in the RAM.

Since the linker must "bind" the program to the absolute hardware addresses of the interface, the compiler must be notified of these addresses. This is done in the file " ProgNr.inc" in the StartupCode folder. Here, the memory area, for which the program is generated, can be sufficiently indicated by FLASH1, FLASH2 or RAM1. FLASH1 is always used in the examples.

Note!
The FtLoader program reads out the current memory address of the interface, which is connected, and displays the values. In future firmware versions of the interface, it could be possible that something on the addresses changes and these can then be changed in the "ncrt0.a30" file.

That, which is normally not possible for embedded programs, is allowed here: "main()" may end itself and then returns to the interface firmware! When main() returns to the firmware, the firmware expects a return code. Therefore, the main() routine should be declared as a function.
The red error LED in the interface can be switched by the return code. A value of 1 to 5 corresponds to the number of blinks. For a value greater than five, the red LED blinks continuously until the PROG key is pressed. For a value of "0," it does not blink.

9 Low Level Programing

With the C Compiler, it is possible to generate programs, which can damage the processor in the interface! In contrast to programs for the fischertechnik RoboPro software, direct access to the hardware ports of the processor can be attained using your own C program. Since for this modern processor, a large part of the internal hardware is configured by software instructions, it is possible that if incorrect settings are made then the processor will no longer work properly with the remaining interface hardware and unfortunately damage could occur due to this. In particular for access to memory through markers, there is the big danger that the processor register can be reached through this.

Therefore, the manufacturer must reject warranty claims for damaged processors due to defective C programs.

10 The Transfer Area

The querying of the inputs and control of the outputs is done through a "transfer area" (communication memory area). This area is matched with the hardware every 10 ms through the interface firmware after the program is started. Programers who have already worked with the FtLib are familiar with this principle. The structure of the transfer area is found in the folder, TA_Firmware, in the header file "TAF_00D.h." The transfer area is identical to the FtLib version. The start address for the transfer area is fixed in the firmware at 0x400 in the RAM.

So that this can be used in a C program, both of the following lines must be at the start of the program:

```
#include    "TA_Firmware\TAF_00D.h"
#include    "TA_Firmware\TAF_00P.h"
```

In the header file "TAF_00D.h," there is a definition of the "transfer area," a data structure, through which the program "communicates" with the hardware in the interface.

In the header file "TAF_00P.h," there is a definition of the firmware macros to call up functions within the firmware.

Both of these files must therefore be contained in every program. In addition, the file "TAF_00.c" must be added to every program (see examples).

10.1 Digital Inputs E1-E32

The bits for the digital inputs are set at "0" for an open input and for an input, which is connected with "+", to "1." Inputs, which are not available, (lack of expansion modules) are set at "0." In addition, all 32 inputs are again stored starting with Base+0x100 in a 16 bit variable per input (1 = input operated).

10.2 Special Inputs

The 11 buttons of the IR remote control are special inputs. The number of the button pressed on the IR transmitter and, the information about if code "1" or code "2" was activated, is stored in the memory position Base+0x0E. In addition, all buttons are stored again in 16 bit variables (as for digital inputs).

10.3 Analog Inputs

The analog inputs are stored as 16 bit values with a value range of 0 to 1023. In addition, the firmware calculates the resistance value (in ohms) for the AX and AY inputs. Therefore, their value range goes from 0 to 5700.

10.4 16 Bit Timer

The six 16 bit timers with increments from 1 ms, 10 ms, 100 ms, 1 s, 10 s and 60 s are freely available. There is no fixed relation between the individual timer values, which means that the 10 ms value is, for example, not 10 times the 1 ms value.

10.5 Outputs

The outputs are controlled by a polarity bit, an energy saving bit and a PWM value byte. The PWM value and the polarity bit are given per individual output. The energy saving bit is given per output pair. If the polarity bit is "0," the output is set to ground and if the polarity bit is "1" then the output is set to power (9V). If the energy saving bit is "1" then an output pair is switched high resistance after a delay (1 sec.) if both associated polarity bits are set to "0". If the energy saving bit is set to "0" then the associated output pair is not high resistance switched. Through the PWM byte, which has a value range of 0 to 7, the pulse width of the output is set in eight steps, for example, in 12.5 percent steps between 12.5. percent and 100 percent.

Instructions for Download Programs
The output settings are copied every 10 ms by the firmware in a separate data area and the PWM values are calculated if the bit "UpdateOutputs" (UA1) is set to BASE+0xE1. If UA2 is set instead of UA1 then the outputs are only written one time for the next 10 ms interrupt. After the outputs are set, the UA2 bit is erased.

After the initialization of the interface (turning on), all energy saving bits are set to "1". This functionality is activated as a standard.

10.6 Installed Expansions Mode

Starting with Base+0xE6, the number of the installed I/O extension modules is found.

11 Calling Up Firmware

In addition to the transfer area, the interface firmware offers some functions, for example, sending a message. These are described in the following.
The callup is done through macros, which are defined in the file "TAF_00P.h" in the folder "TA_Firmware".

11.1 void SetFtDeviceReset (char mode)

```
mode:   0 =   Cold start / RESET
              (turn on like interface)

        1 =   PROG end
              (init, without RAM erasing)
```

This procedure stops the application program. When the callup is made, it can be done so that the interface makes a complete "Restart" (hardware reinitialized, erases RAM) or only the hardware and the firmware are reinitialized without deleting the RAM.

11.2 void SetFt1msTimerTickAddress(void far *())

With this function, the application program notifies the firmware of the address of the timer routine. After this, this is called up every 1 ms. The functionality can be turned off with the value ZERO.

11.3 void SetFtDeviceCommMode (BYTE mode, BYTE value1, UINT *pData)

This routine sets the mode of the serial interface in the interface. After it is turned on, the interface is in the normal mode. In this mode, the interface can be controlled in the online mode.
In the "Message Mode", messages can be sent from one interface to another through the serial interface.

The mode, which is set, remains active until a new mode is set with this function. By pressing the "Port" button on the interface, the mode IF_COM_ONLINE is reset if the interface is in the AutoScan mode.

The mode, which is currently set, can be queried through "IF_COM_PARAMETER". The result is stored after the transferred address, to which the "pData" points.

Modes

```
Mode =    IF_COM_ONLINE:     Set standard mode
                             The interface is set to the standard mode.
                             (Parameters: 38400,n,8,1).

Mode =    IF_COM_MESSAGE:    Messages sent through serial
                             interface.

Mode =    IF_COM_PARAMETER   Read out mode
                             Result:
                             pData: Highbyte = Value if mode = Data
                                    Lowbyte =  Mode
```

11.4 void GetRfParameter (BYTE *pData)

This function reads out the parameters of the RF module and stores these after the address, to which the pData points. The data field must have a size of at least 4 bytes.

```
pData[0]  Error code
          ERROR_SUCCESS: No error
pData[1]  Mode
          0=Module turned off
          1=Module active
pData[2]  Frequency
pData[3]  Module number (radio call number)
```

11.5 void ClearFtMessageBuffer (void)

This function erases the message buffer in the interface.

11.6 void SetFtMessageReceiveAddress (void *())

With this function, the application program notifies the firmware of the address of the receive message function. If p=0 is transferred, then no function is called up upon the receipt of a message by the firmware. Then this function is called up for every incoming message. The interface firmware then transfers a "near" marker to the message received.

11.7 BYTE SendFtMessage (BYTE hwid, BYTE subid, ULONG message, UINT uiWaitTime, UINT uiOption)

This function sends a message. The actual message consists of a 16 bit message ID in Lowword of the parameter "message" and a 16 bit message value in Highword of the parameter "message". A physical channel is selected with the parameter "hwid".

```
MSG_HWID_SELF     (0)  Send message to self

MSG_HWID_SER      (1)  Send message to serial interface

MSG_HWID_RF       (2)  Distribute message over radio
                       (but do not send to self)

MSG_HWID_RF_SELF  (3)  Distribute message over RF
                       (also to self)
```

Through the subchannel ID, a physical channel can be divided into several logical channels. IDs from 0 tof 219 are permitted. The values from 220 to 255 are reserved for internal tasks.

The return value is "0" in case of success, otherwise an error number. If there is still space in the buffer, the function returns immediately, otherwise the function waits at the maximum for the time in ms indicated in "uiWaitTime."

The message system requires about 5 ms for the distribution of a message. Since in particular within loops, this function could be called up significantly more often, the number of messages to be sent can be optimized through the parameter "uiOption". With this you can prevent the same message from being sent several times.

If messages are also to be sent to the serial interface of the interface (bHwId = MSG_HWID_SER) then the mode IF_COM_MESSAGE must be activated before the start of the transfer thread with the function " SetFtDeviceCommMode()".

Return values

ERROR_SUCCESS	**No error**
FTLIB_ERR_MSG_BUFFER_FULL_TIMEOUT	**TimeOut, message could not sent within the indicated time**

Call:	BYTE hwId	Hardware-ID
	MSG_HWID_SELF (0x00):	Copied direct into the own receive buffer
	MSG_HWID_SER (0x01):	Sent through interface RS232
	MSG_HWID_RF (0x02):	Over radio only to other modules
	MSG_HWID_RF_SELF (0x03):	Over radio also to self
	BYTE subId	Logical channel, the IDs are permitted 0 to 219. The values from 220 to 255 are reserved for internal tasks
	ULONG message	Lowword: 16 bit message ID Highword: 16 bit message
	UINT uiWaitTime	If the internal buffer is full, this parameter (in ms) can be used to set how long to wait until the function returns.
	UINT uiOption	Transmission options
	MSG_SEND_NORMAL (0):	The message is written directly into the transmit buffer
	MSG_SEND_OTHER_THAN_LAST (1):	The message is not sent if an identical message (bHwId, bSubId, dwMessage) is at the end of the buffer. If the buffer is empty or a different message is at the end of the buffer then the message is sent.
	MSG_SEND_IF_NOT_PRESENT (2):	The message is not sent if an identical message (bHwId, bSubId, dwMessage) is somewhere in the buffer. If the buffer is empty, then the message is sent.
Return:	Error-Code (in R0L)	
	ERROR_SUCCESS	No error
	FTLIB_ERR_MSG_BUFFER_FULL_TIMEOUT	Buffer is full, message could not be tranmitted in the time given.

11.8 void FtDelay (UINT)

This procedure waits for the time given (in ms).

11.9 void SetFtDistanceSensorMode (UCHAR ucMode, UCHAR ucTol1, UCHAR ucTol2, UINT uiLevel1, UINT uiLevel2, UCHAR ucRepeat1, UCHAR ucRepeat2)

This routine initializes the D1/D2 input on the interface for the connection of the fischertechnik distance sensor or for the measurement of voltages in the range of 0 to 10 volts.

Important Information
Since the mode of the D1 / D2 inputs can be set with software, we recommend that no voltages be "directly" fed into these connections in order to avoid damage to the interface due to software errors. Since the inputs are high resistance, a resistance of about 220 ohms to 470 ohms should be connected directly to the D1 / D2 jack (series connection). We recommend that the voltage to be measure be only connected "after this."

Call:	FT_HANDLE hFt	Handle of the device
	DWORD dwMode	Mode of the connections: IF_DS_INPUT_VOLTAGE (0x00) = inputs measure voltages IF_DS_INPUT_DISTANCE (0x01) = Inputs for ft distance sensor

The following parameters are depedendt on the mode, which is set.
For dwMode = IF_DS_INPUT_DISTANCE the following applies:

DWORD dwTol1	Tolerance range D1 Recommended: IF_DS_INPUT_TOL_STD (20)
DWORD dwTol2	Tolerance range D2 Recommended: IF_DS_INPUT_TOL_STD (20)
DWORD dwLevel1	Threshold value D1
DWORD dwLevel2	Threshold value D2
DWORD dwRepeat1	Repeat value D1 Recommended: IF_DS_INPUT_REP_STD (3)
DWORD dwRepeat2	Repeat value D2 Recommended: IF_DS_INPUT_REP_STD (3)

The distance sensor works with infrared light and can therefore experience interference from outside influences
such as IR hand transmitters. In order to prevent this interference, the provision of the repeat value can be used to set how often the "same" measured value must measured until this is considered to be valid. Since the measured values may fluctuate from one measurement to the next, there is a tolerance range. As soon as a new measurement starts, the following measured values may change within this "window" without causing a restart of the measurements. The threshhold value determines the level for the evaluation as "digital" distance sensors. Below the threshhold value, a logical "0" is sent and above this a "1" is sent. In the transfer area, the conditions determined are stored by the firmware in the memory positions "Base+0x0C" (digital) and "Base+0x1C / Base+0x1E" (analog).

12 Communication

Several Robo Interfaces can exchange messages with each other through the RS232 interface and the radio interface. Basically, the messages consist of a 16 bit message ID and a 16 bit value.

The communication is done on radio channels (frequencies). A maximum of eight radio interfaces can exchange messages on the same radio channel. For this, every module has its own "ID" number, which can be set in the range from 1 to 8. The PC module always has the ID number "0". In addition, messages can be sent to the serial interface of the interface and received by this serial interface.

Every radio channel is divided into 256 logical subchannels, which serve for the structuring of the communication. For this, a message is always distributed to all participants ("broadcasting"). Every message consists of five data bytes:

SubId	Number of the subchannel (1 byte)
Message	Message data (4 bytes)
	B1:B0: Message ID (Low-Word)
	B3:B2: Message (High-Word)

There is no feedback to the "Sender" to confirm that the message was received.

The communication is controlled by the PC module, which works as a message router.

12.1 Serial Messages

C programs can also send and receive messages through the serial interface. Every message is sent individually through the serial interface. For this, the mode in the interface must be changed with the function "SetFtDeviceCommMode()". The changed mode is then indicated by the continual illumination of the COM LED on the interface. By pressing the "Port" button, the mode is reset, but of course this can be done through the software as well. It is also possible to exchange messages between two Robo Interfaces through the serial interface. The proper connection cable for this (X cable) can be obtained from the fischertechnik Individual Parts Service.

12.2 Firmware Support

The interface firmware offers the function "SendFtMessage()" for the data transport. When sending, a physical channel can be selected using a hardware ID. If no second interface or no radio interface is available, you can send the message "to yourself." You can send the message to the serial interface or have it distributed by radio (RF) to other modules by the message router.

12.3 Receiving Messages

As soon as the interface receives a message, it calls up a callback routine and transfers a "near" marker to this for the message received. Using the function "SetFtMessageReceiveAddress()", the address of this message processing function can be transferred to the firmware. There is an example of this in the scope of delivery of the C Compiler. The message processing function stores the message in a buffer of the C program. The C program then picks up the message from the cache memory at a later time.

13 Debugging

The debugging of programs in the interfaces is not possible at this time because Renesas does not publish the control codes for the debugger interface for its tools.

As an alternative, the memory areas in the interface RAM can be read out using the FtLoader. The variables "ucDbg1F0" … "ucDbg1FF" (Base+0x1F0..0x1FF) are avaible for this in the transfer area. The setting of values in the RAM is also possible after a program has been started with the FtLoader.

14 Revision

Version 0.65:	Complete revision
	-Renaming ClearFtMessagePuffer() to ClearFtMessageBuffer()
Version 1.66a	-Renaming of 0.65 to 1.66a
	-Revision of the storage layout
	-Support of the interface firmware 01.66.00.03