To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.

2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.

4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.

6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.

8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.

10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

C Compiler Package for M32C/90,/80, M16C/80,/70 Series
V.5.41 Release 01
Guidebook
(Rev.1.0)

Renesas Solutions Corporation
Dec 16, 2006

Abstract

This guidebook describes how to use the C Compiler Package for the M32C/90/80 and M16C/80/70 series when you have introduced it in your development work. Please refer to this guidebook when you install the C Compiler Package and when you create a project, etc.

1. Installation guide of C compiler package

1.1. Before installing C Compiler Package

Please confirm as follows before installing C Compiler Package in your computer.

(1) Please carefully read the "License Agreement" and "Release Note" included with your product before using C Compiler Package. If you've installed this product in your computer, it is assumed that you've agreed to the provisions stipulated in the License Agreement.

(2) In order that C Compiler Package operates comfortably, it requires at least 32Mbytes of memory and a harddisk having 20Mbytes or more of space.

(3) You need to input a license ID in the middle of installation. Before you start installing C Compiler Package, check your license ID.

1.2. Required System Configuration

Host Computer	IBM [1] PC/AT compatible
CPU	Pentium 4 (or Pentium M) or more is recommended.
OS	Windows [2] 98 Windows Me Windows NT 4.0 Windows 2000 and Windows XP
Memory	512MB or more is recommended.
Capacity of hard disk	200MB or more in empty capacity
Display	Resolution of SVGA or more
I/O device	CD-ROM drive
Else	Pointing device of mouse etc.

1.3. Installation Procedure

Insert the CD-ROM of the C Compiler Package into the CD drive of your computer, and High-performance Embedded Workshop Install Manager will start up automatically. Follow the messages displayed by High-performance Embedded Workshop Install Manager as you install the C Compiler Package.

If High-performance Embedded Workshop Install Manager does not start up automatically, execute "HewInstMan.exe" included in the CD-ROM.

```
D:¥> HewInstMan.exe

The drive name "D:" differs with each PC used.
```

Note, however, that before High-performance Embedded Workshop Install Manager starts up, all other applications must be closed.

● Precautions

[1] Make sure that High-performance Embedded Workshop V.4.01 is installed in a different directory than the one where the earlier version V.1.x is installed.

[2] The projects created with High-performance Embedded Workshop V.1.0x cannot be used directly in V.4.01. To use any project created with V.1.0x in V.4.01, convert it to V.1.1x once (by opening a project in V.1.1 and then saving it).

The following describes the procedure for newly installing the C Compiler Package in your computer while High-performance Embedded Workshop Install Manager is up and running.

---

[1] IBM and AT are registered trademarks of International Business Machines Corporation.
[2] Windows(R), and Windows NT(R) are registered trademarks of Microsoft Corporation in the U.S. and other countries.
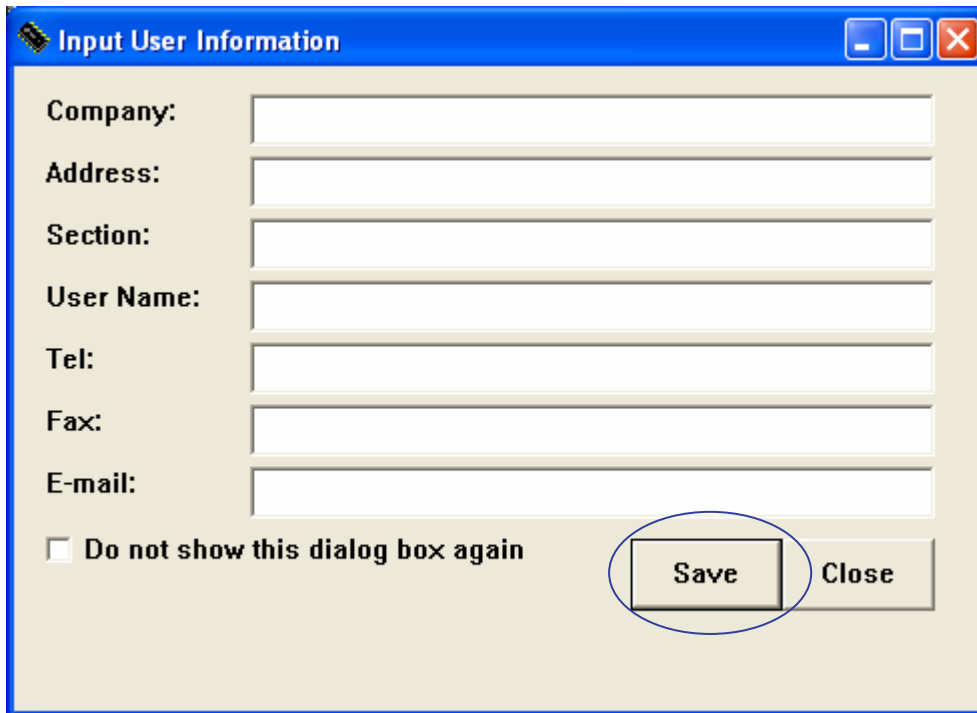
(1) When you are newly installing the C Compiler Package or when you click "Read first" in High-performance Embedded Workshop Install Manager, the Install Manager Help shown below is displayed on the screen. Please read it before you proceed.
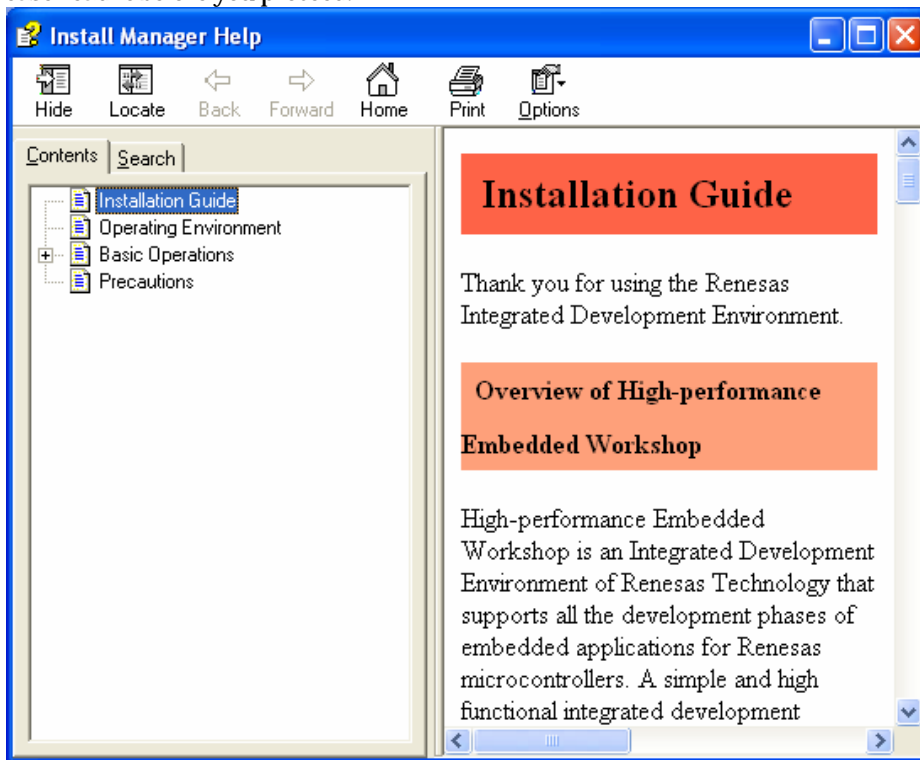
(2) Click the Installation button.
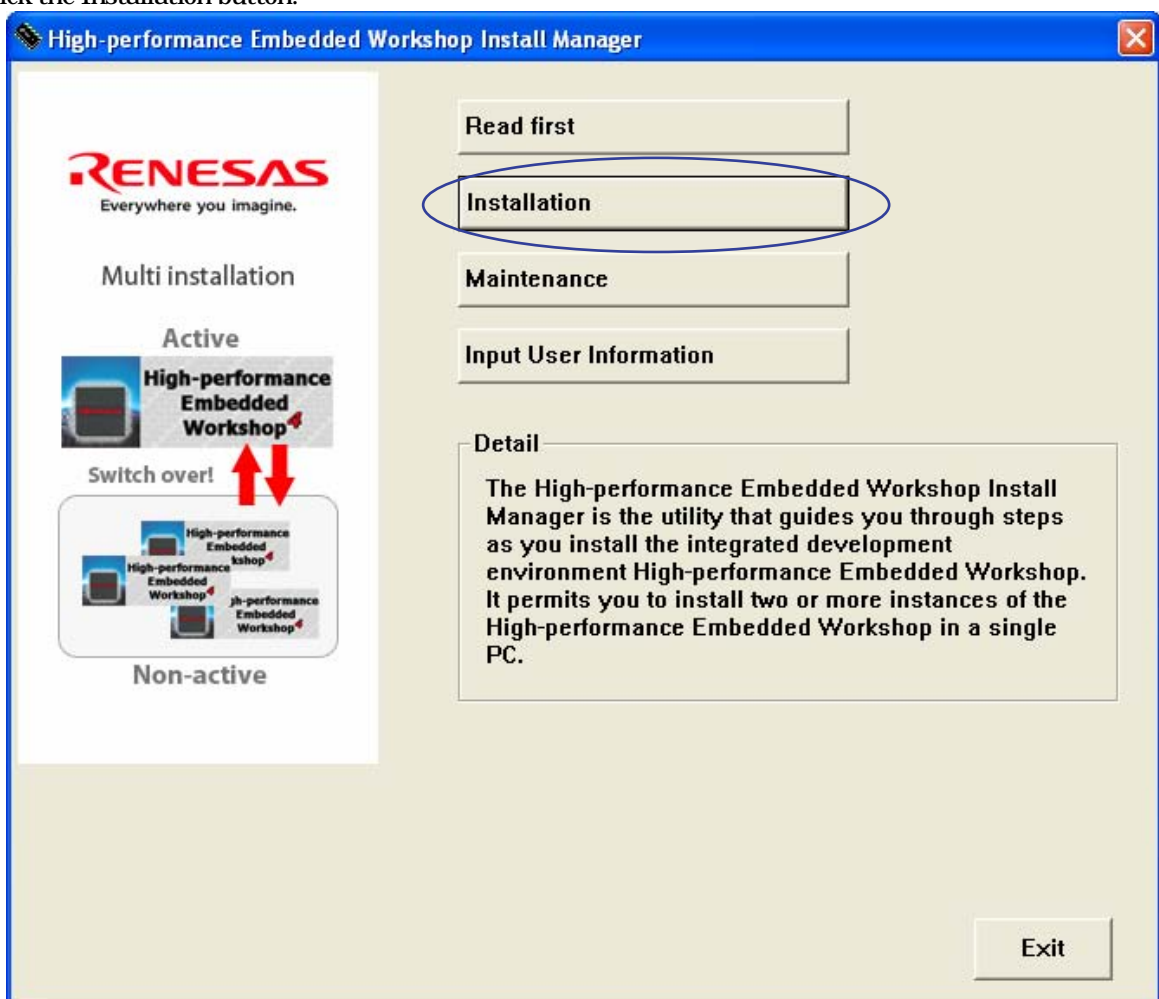
(3) Enter the user information and click Save.
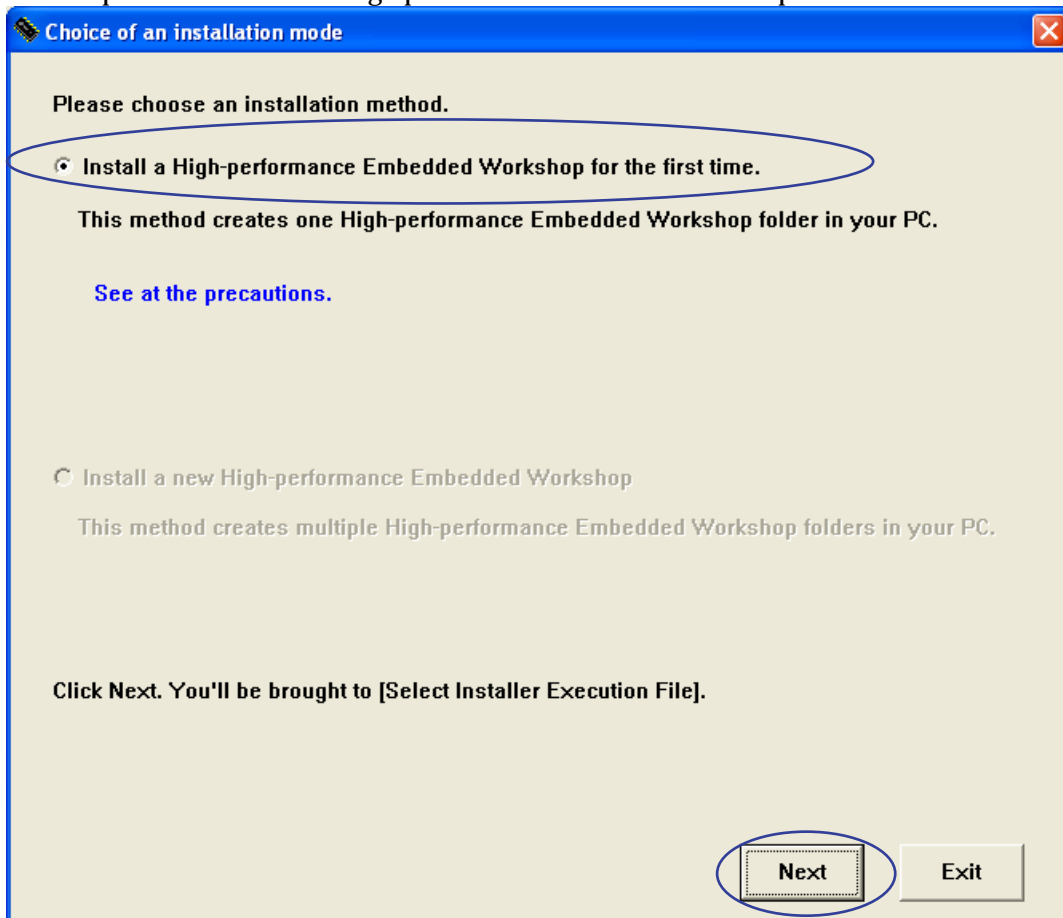
(4) Select the option titled "Install a High-performance Embedded Workshop for the first time" and click Next.

(5) Check the folder in which the High-performance Embedded Workshop will be installed and click Next.

Choice of an installation

Choice of an installation

Choose a folder to install.

When you install it in this folder , please click a [Next] button.
When you install it in another folder , click a [Change] button, and please choose a folder.

Folder of an installation

F:\Program Files\Renesas\Hew   Change

Back   Next   Exit
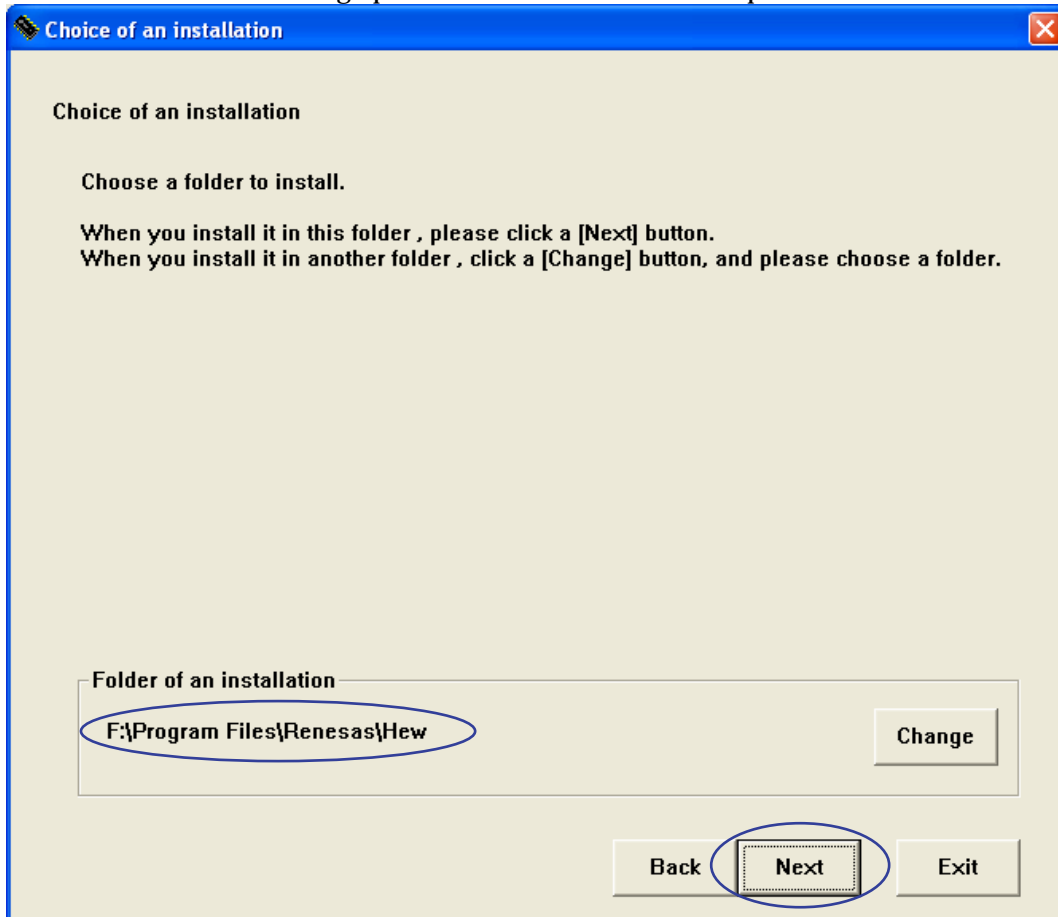
(6) Select all check boxes and click Install.

Choice of an installation product

Please choose a software product to install.

☑ M32C Toolchains V.5.41 Release 01
☑ AutoUpdate

Detail

Please choose from the list (it will installed one-by-one by this order:

Please push a [Install] button after having chosen a software product to install.It execute installation program.

Back   Install   Exit
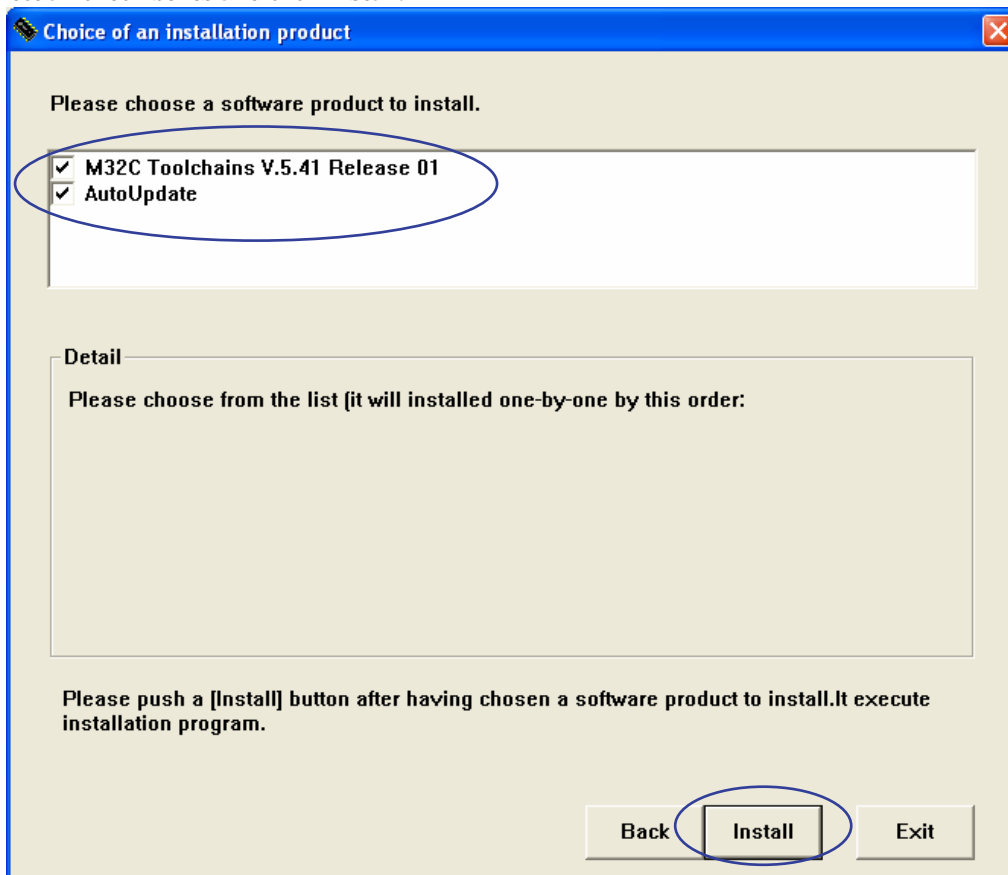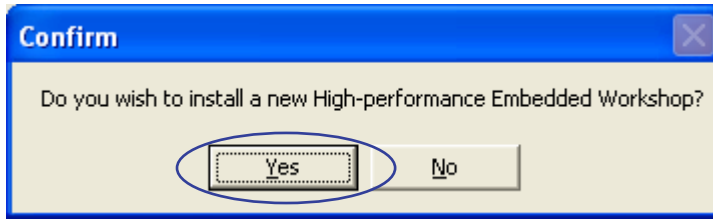
(7) Click Yes.

(8) Select English and click Next.

(9) Click Next.
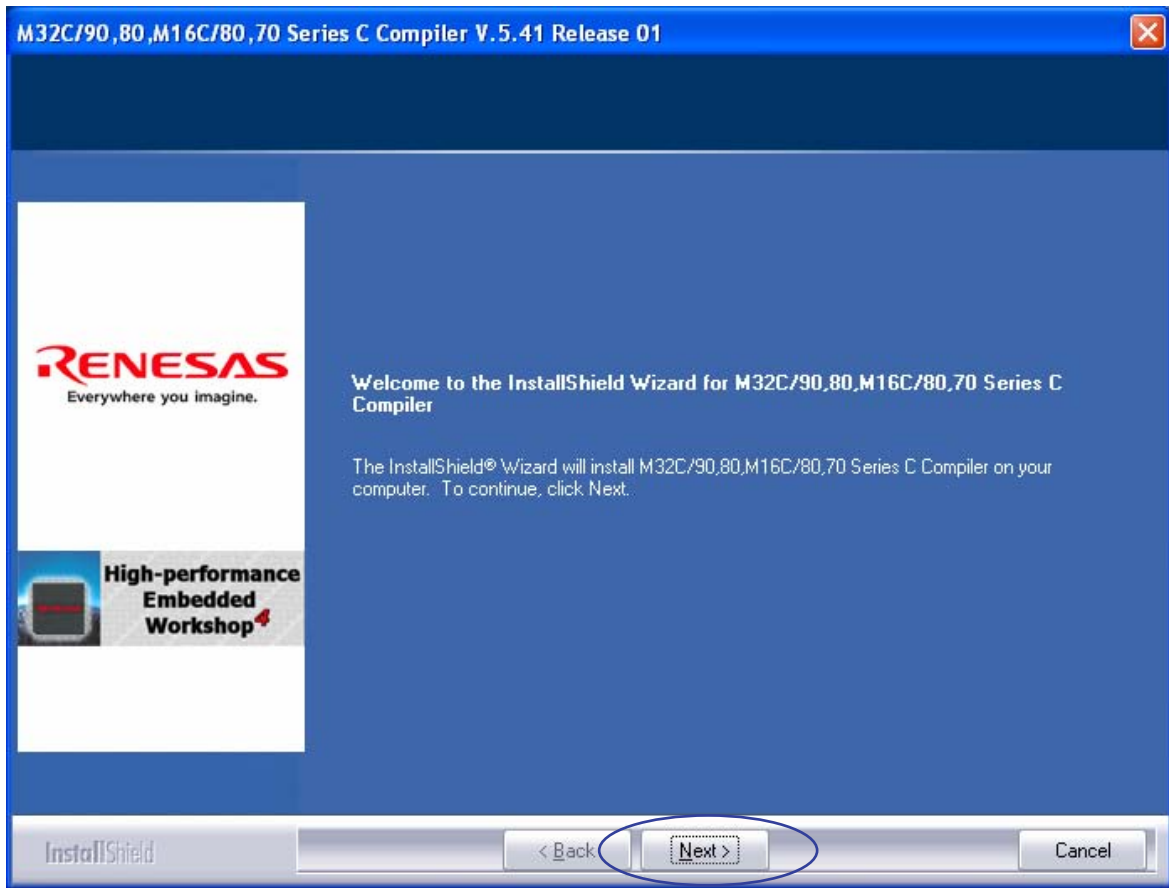
(10) Carefully read the Software User License Agreement and click Yes.
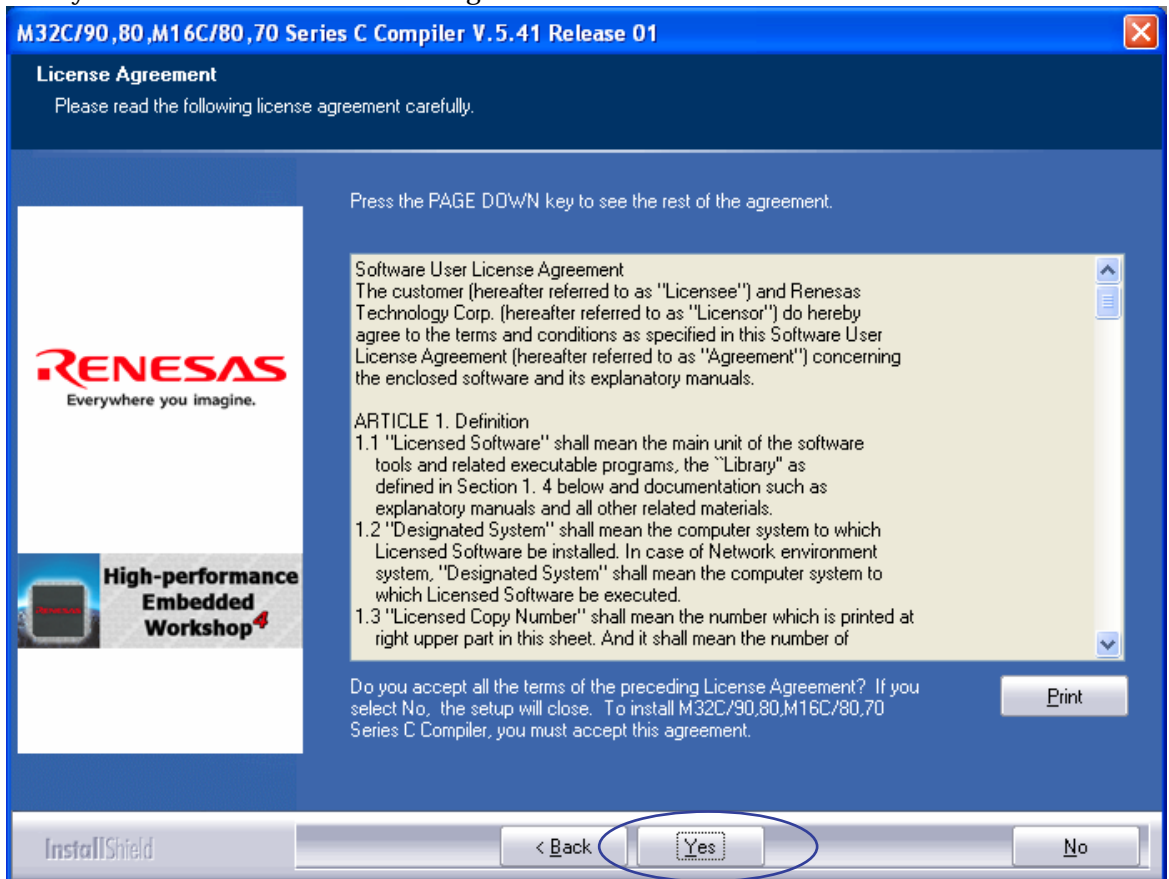
(11) Enter the license ID written in your license ID certificate and click Next.
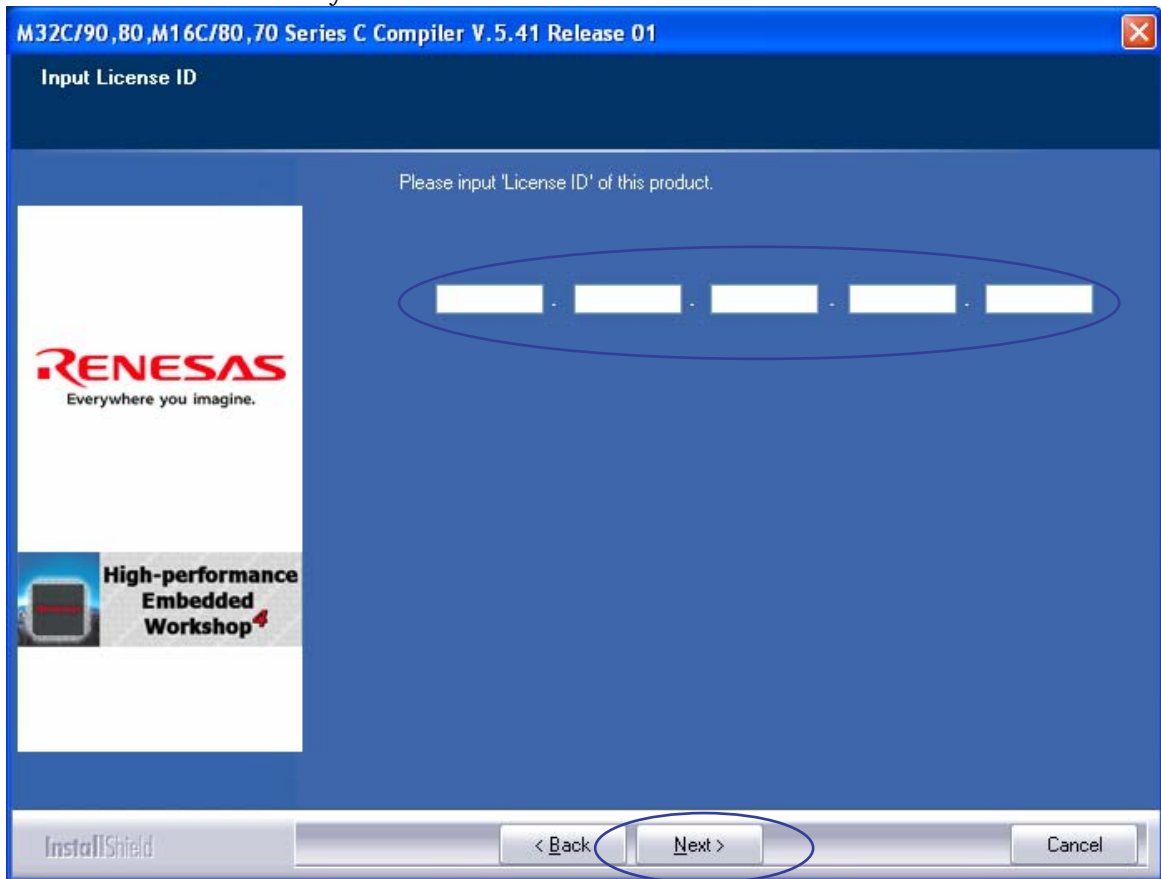
(12) Check the folder in which the C compiler will be installed and click Next.
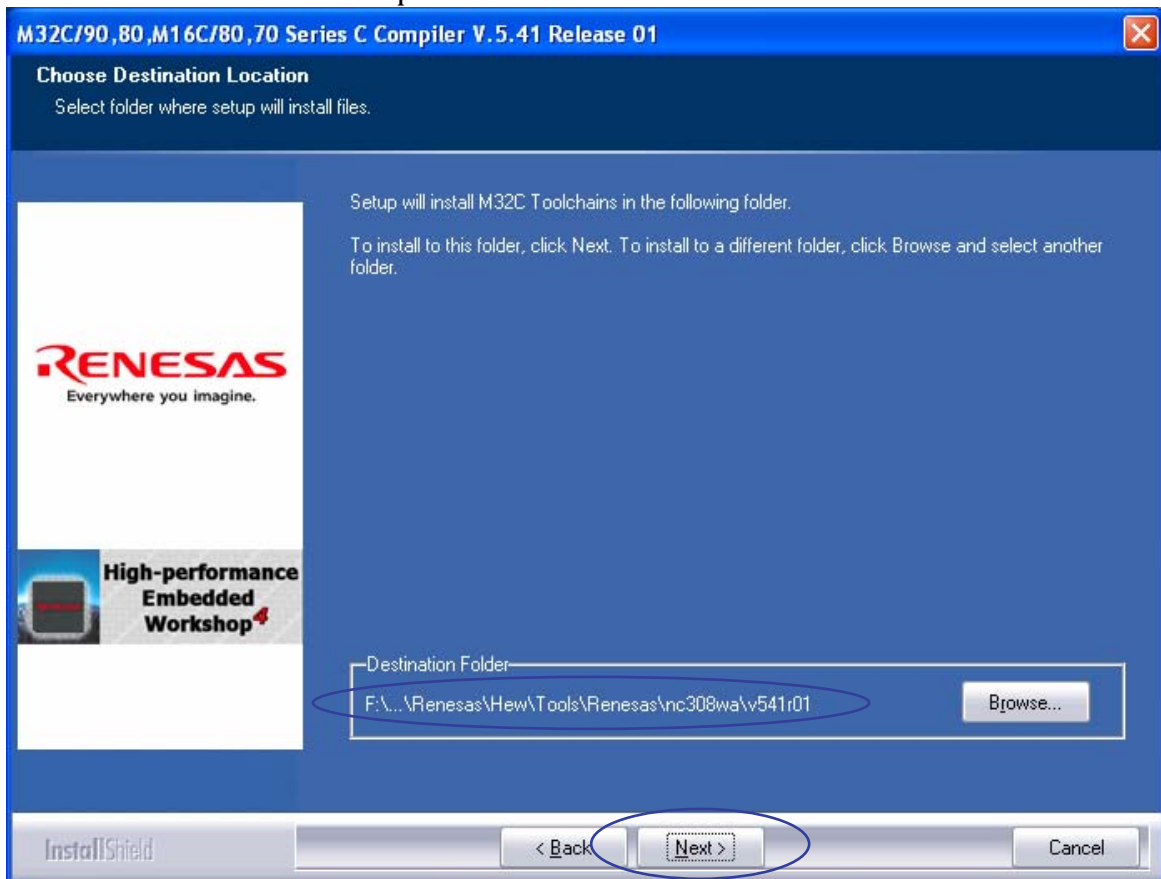
(13) Click Next.
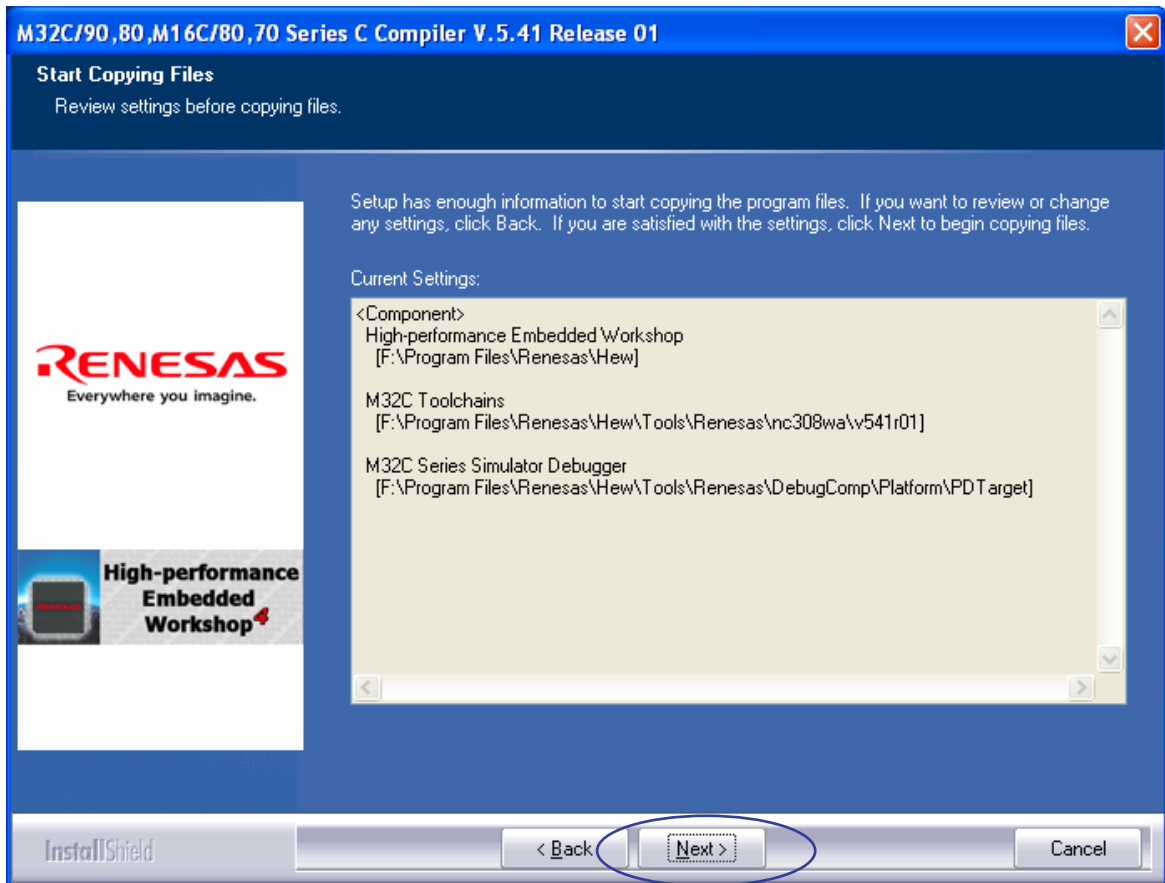
(14) Select the "Executes a Support Information Tool" check box and click Finish.

(15) Click Yes.

Renesas Support-information

⚠️ Some of user information have not been input yet.
Could you fill all of them ?

[Yes]   No
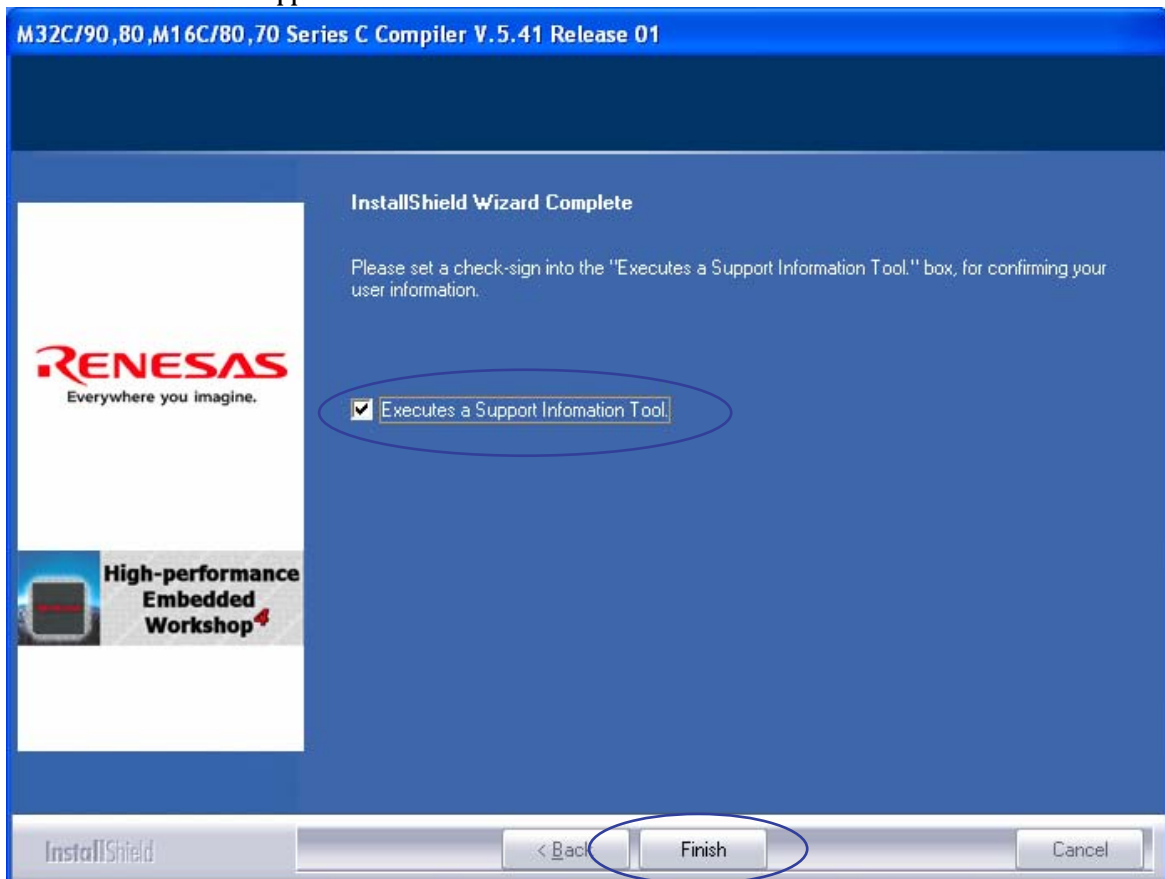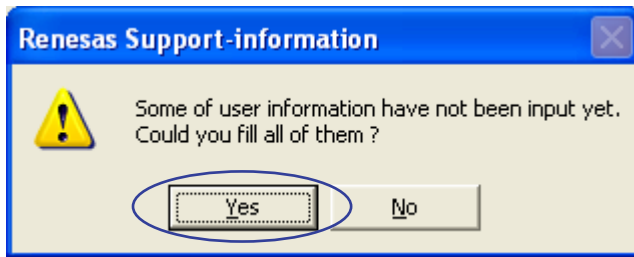
(16) Enter the name of your country in the Country text box and click Save.

Enter Your Profile

Company Name * :	Renesas Solutions Corporation
Company Address * :	Nippon Bldg., 2-6-2 Otemachi, Chiyoda-ku, Tokyo, 100-0004
Country * :	
First Name / Family Name * :	Taro Tool
Position / Job Title * :	Tool
Email * :	csc@renesas.com
Telephone * :	+81-3-5201-5380
Fax :	+81-3-5201-5383
PC :	IBM PC (Pentium 4 CPU 1.60GHz)
OS * :	Windows XP (5.10.2600 (SP2))
Memory Size :	767M Bytes

* Indicates Required Fields

[Save]   Cancel

(17) Check the contents displayed in the Renesas Tools Support Information dialog box and click Output Support.txt. If you want to correct the displayed content, click Edit User Information and make corrections.
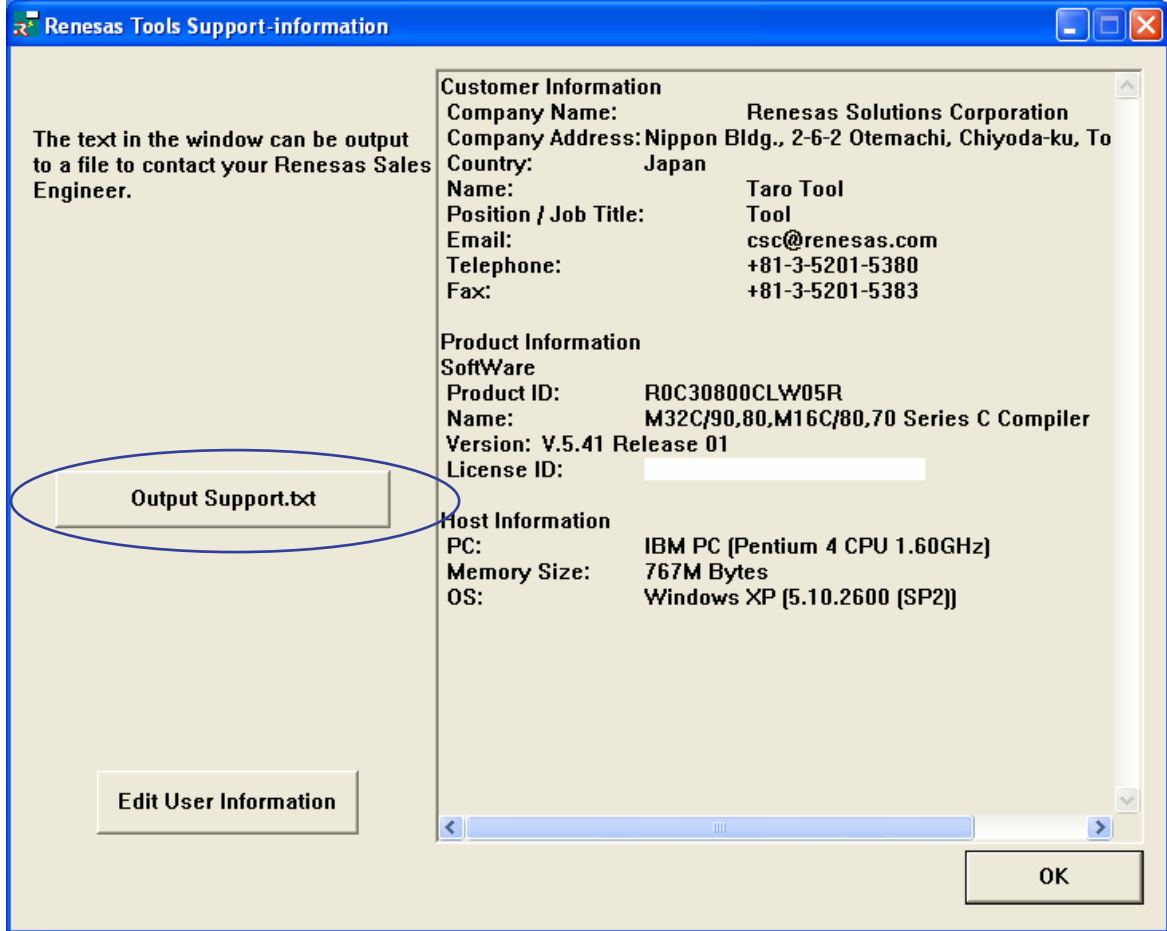
(18) Select any location in which you want the TECHNICAL SUPPORT INFORMATION SHEET file (support.txt) to be created and click Save.
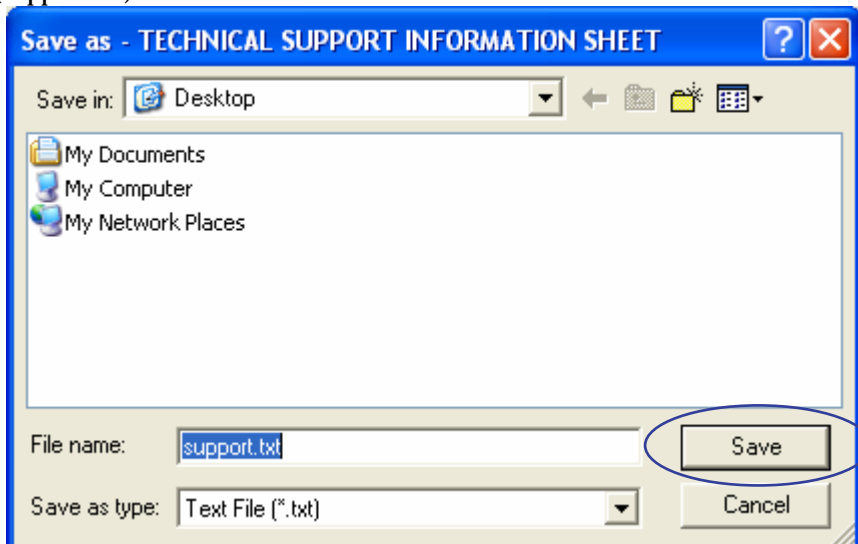
(19) Click OK.
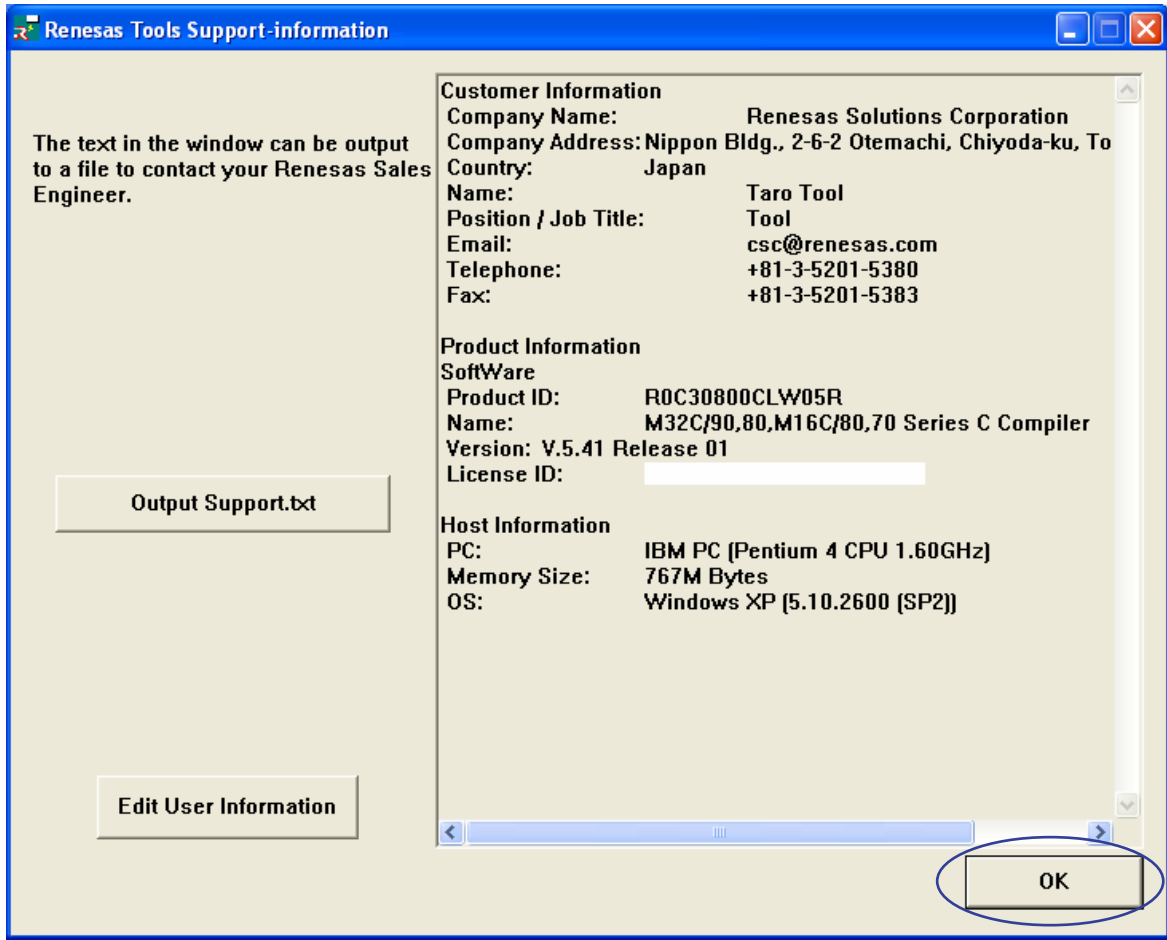
(20) Select English and click Next.
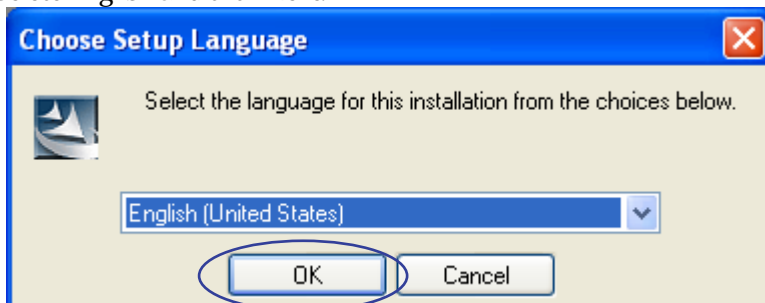
(21) Click Next.

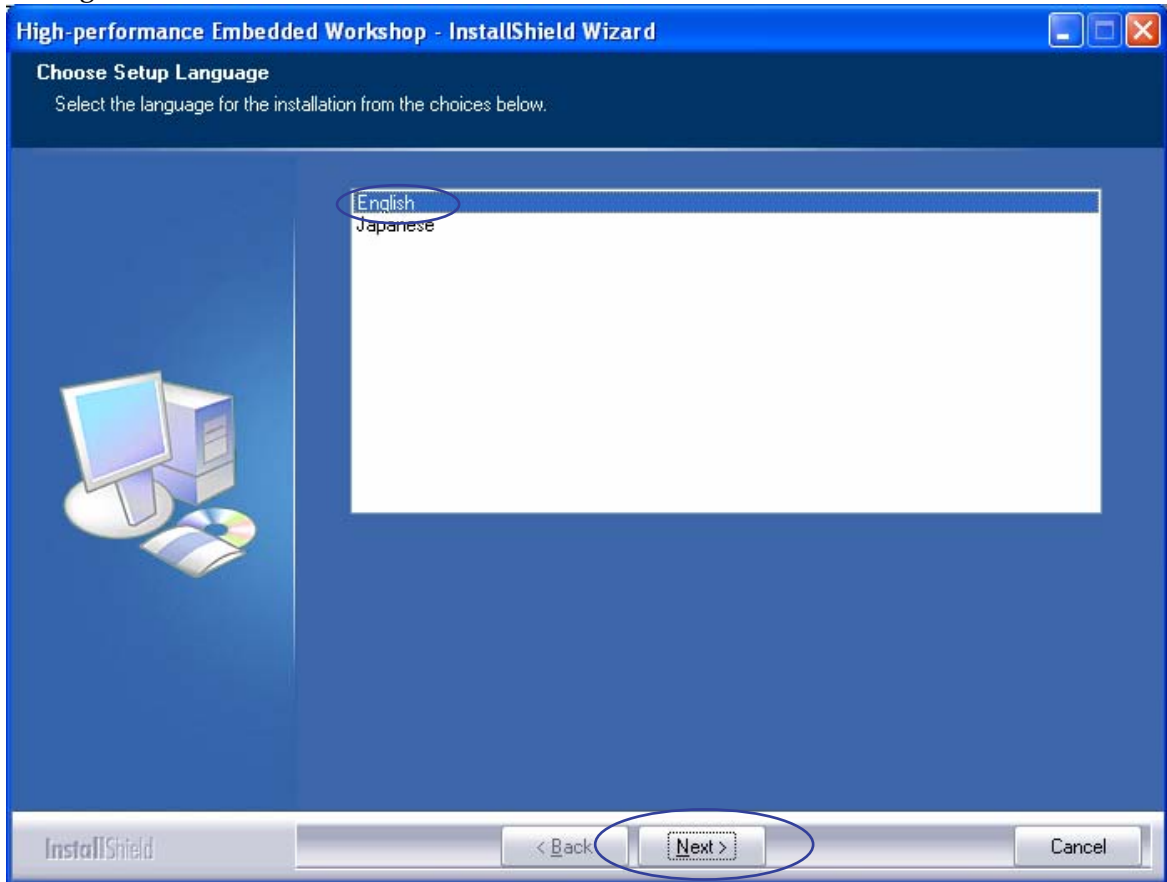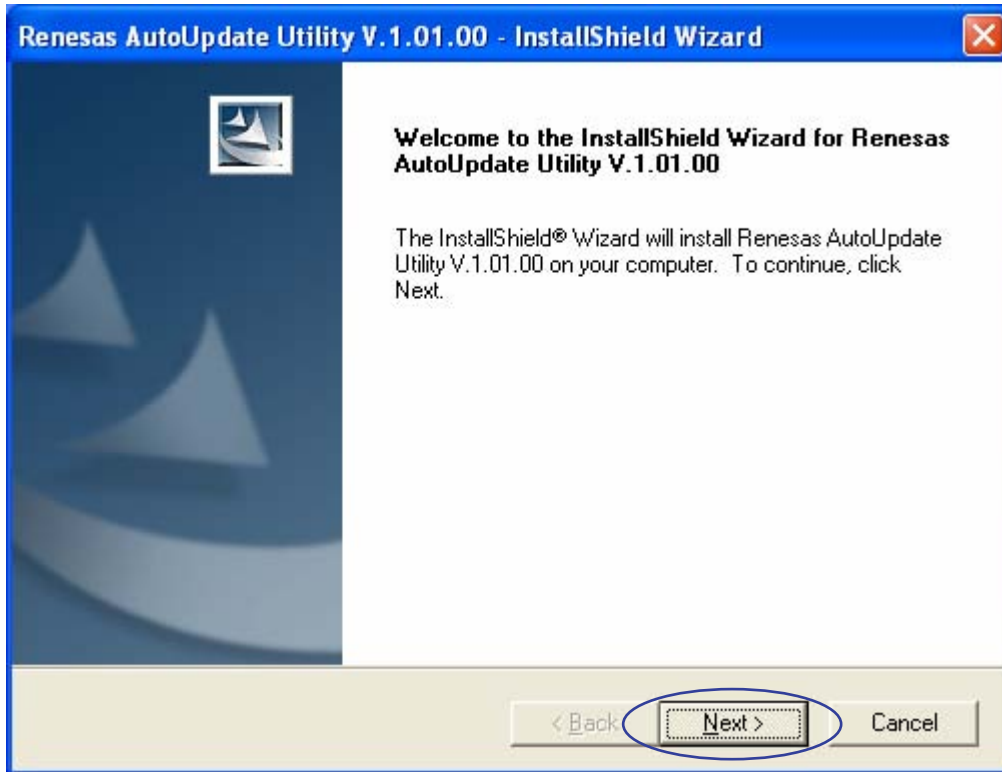(22) Check the folder in which the "I want to register the AutoUpdate utility to the startup folder." will be installed and click Next.
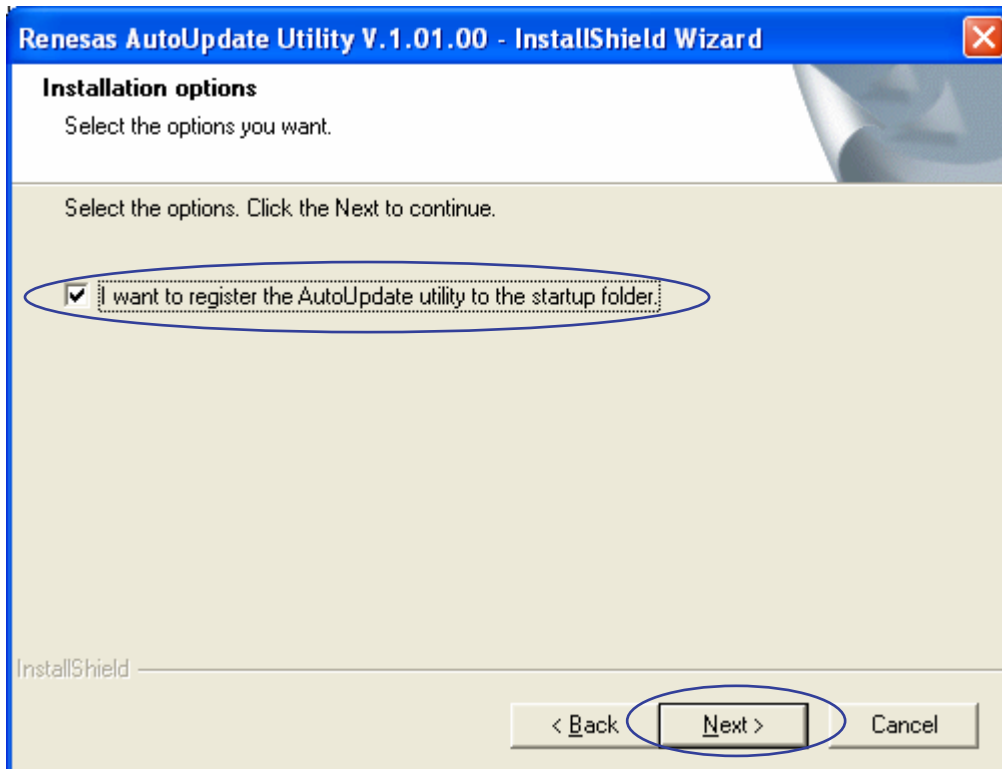
(23) Click Install.
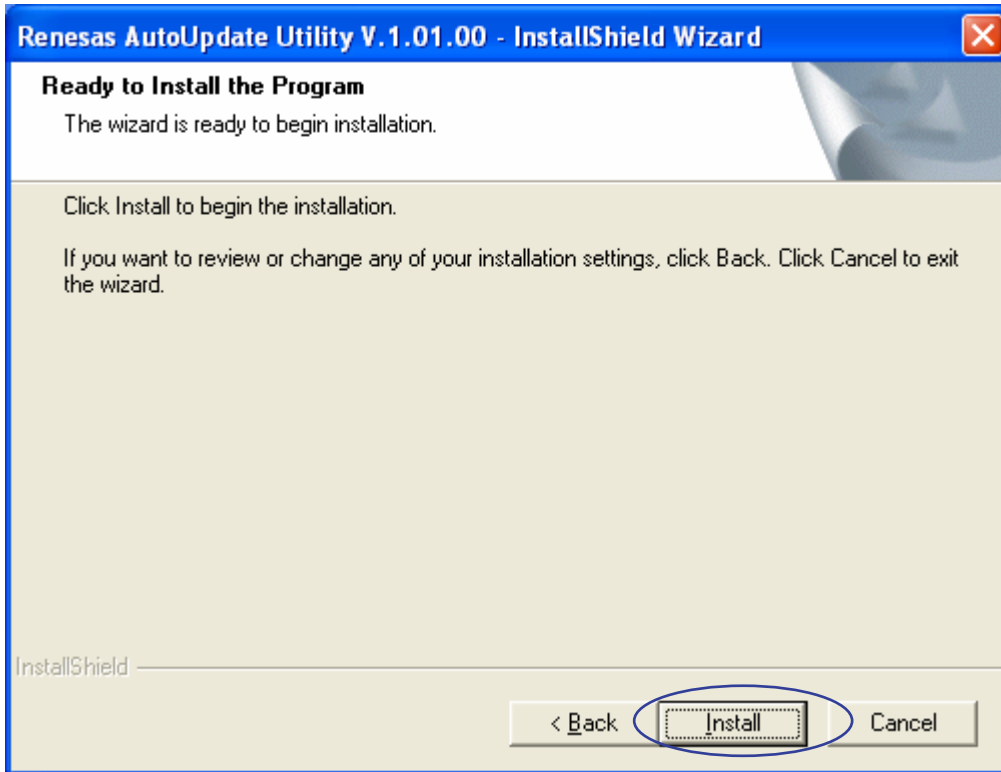
(24) Click Finish.
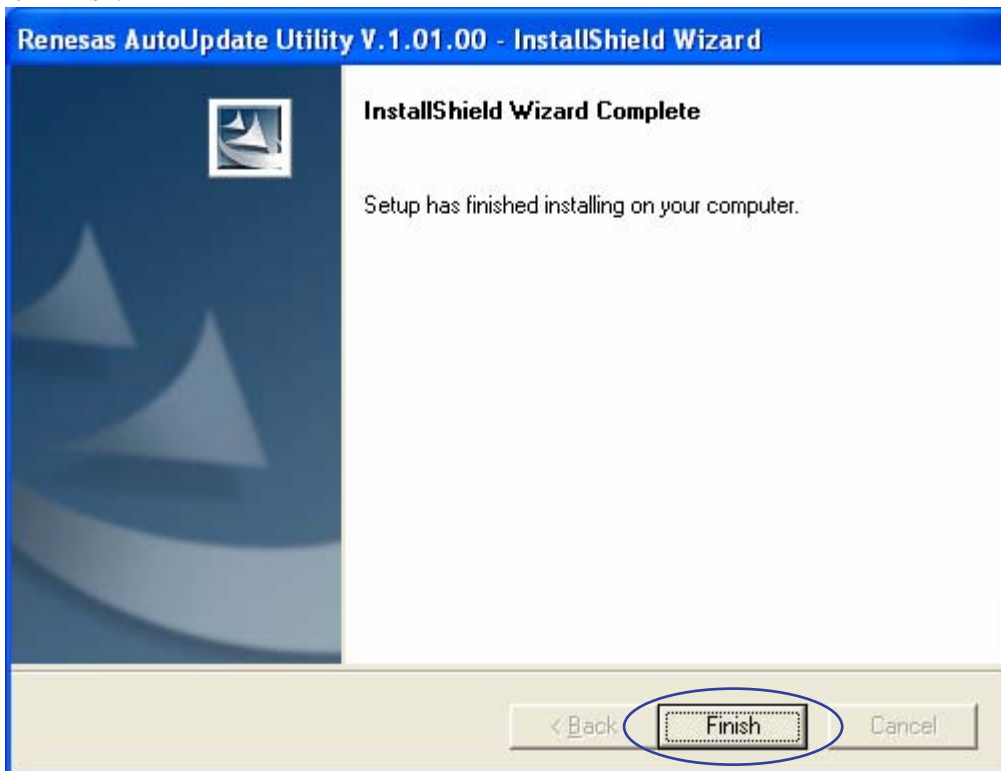
(25) Click Exit to close High-performance Embedded Workshop Install Manager.
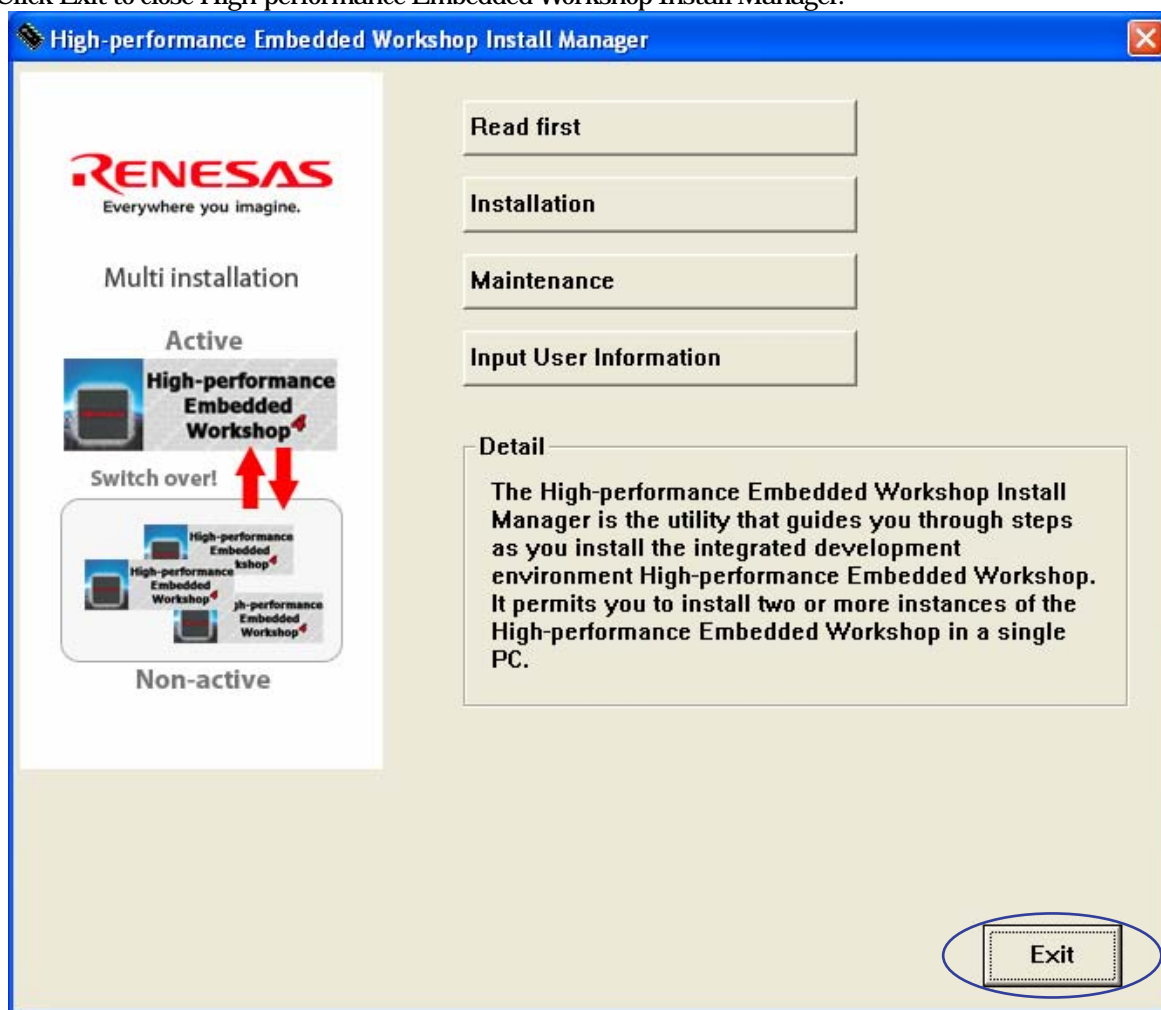
1.4. Uninstalling programs

(1) The installed programs can be uninstalled according to the following instructions. Begin the un-installation after closing all the applications.

(2) Select [Control Panel] on the Windows [Start] menu.

(3) Select the [Add or Remove Programs] icon.

(4) Click [High-performance Embedded Workshop] on the [Change or Remove Programs] tab and click the [Remove] button.

(5) Follow the instruction displayed on the screen.

1.5. Startup or termination of program

1.5.1. Startup and termination of the High-performance Embedded Workshop

● Startup
Click [High-performance Embedded Workshop] in the [High-performance Embedded Workshop] folder in the [Renesas] folder in the [Program] folder of the Windows [Start] menu.

● Termination
Click [Exit] on the [File] menu.

1.5.2. Start of Manual Navigator

- Startup
  Click [Manual Navigator] in the [High-performance Embedded Workshop] folder in the [Renesas] folder in the [Program] folder of the Windows [Start] menu.
- Termination
  Termination: Click [Exit] on the [File] menu.

1.5.3. Displaying the online manuals and attached documents

Displaying the online manuals and attached documents

- Note
  [1] Manual Navigator requires Adobe Reader [3].
  [2] If Manuals folder is moved, Manual Navigator cannot show them.

1.6. Setting when compiler is used on DOS prompt and command prompt

Please execute setnc308.bat when you use the compiler on the DOS prompt and the command prompt.

1.6.1. Environment Variables and Path

Environment variable	Usage
BIN308	Directory in which the C compiler execution files (e.g., *.exe) are stored
INC308	Directory in which the standard include files of the C compiler are stored
LIB308	Directory in which the standard library files of the C compiler are stored
TMP308	Directory in which the temporary files generated by the C compiler are stored
PATH	Directory in which the C compiler generates temporary files Select the directory for which you have access rights.

1.6.2. Batch file

A batch file named "setnc308.bat" will be generated in the directory in which you've installed the C compiler. This file has written in it the environment variables that the C compiler uses.
To use the C compiler from the DOS or the command prompt, execute setnc308.bat.

- Contents written in the batch file

```
REM ***** Environment variable for M32C Toolchains *****
SET BIN308=C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥nc308wa¥v541r01¥BIN
SET LIB308=C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥nc308wa¥v541r01¥LIB308
SET INC308=C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥nc308wa¥v541r01¥INC308
SET TMP308=C:¥Program Files¥Renesas¥Hew¥Tools¥Renesas¥nc308wa¥v541r01¥TMP
SET PATH=%BIN308%;%PATH%
```

---

[3] Adobe and Acrobat are registered trademarks of Adobe Systems Incorporated.

2. Migration to C Compiler Package V.5.41 Release 01

The following describes the precautions to be observed when you upgrade the C Compiler Package from the earlier V.5.20 Release 02 to V.5.41 Release 01.

2.1. Changing Startup Program

Beginning with V.5.40 Release 00, the name of the library function init() has been changed to _init().
Therefore, if you attempt to build without modification, an error message '_init' value i undefined may be generated during a link process.
- Description
   This error occurs when one of the following holds true.
   ◆ You selected "Use Standard I/O Library" when you created projects with V.5.20 Release 02 or earlier.
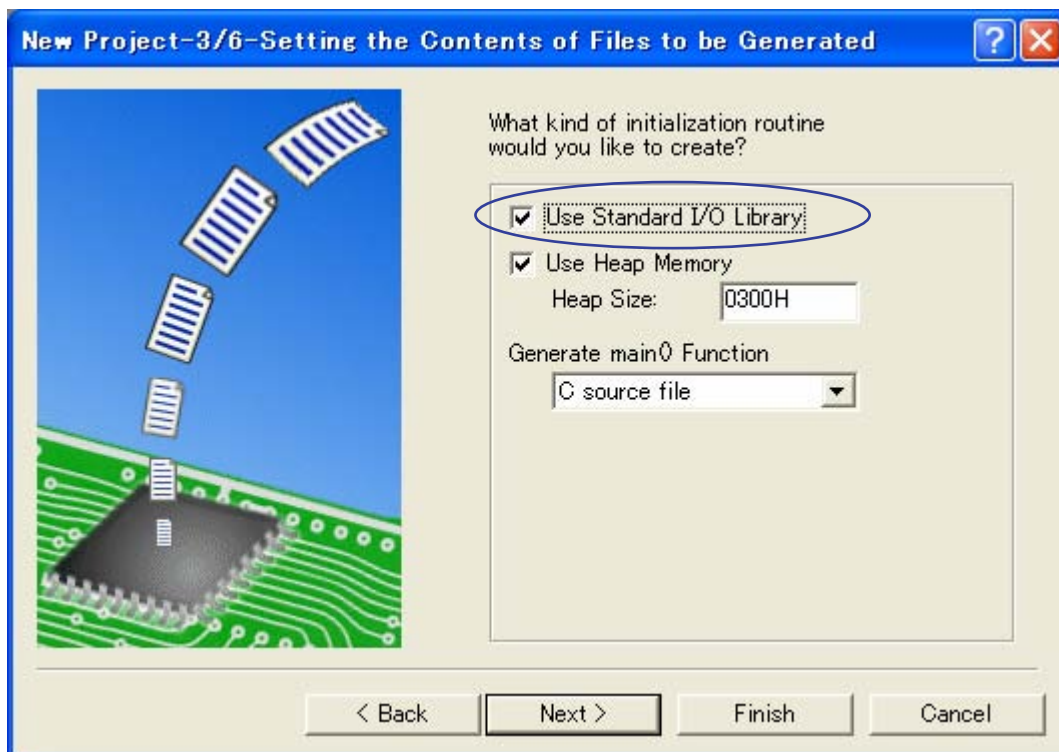
   ◆ The init function is called as when the content of the assembly language startup program "ncrt0.a30" is altered directly.
- Workaround
   ◆ When you are using the startup file (ncrt0.a30) supplied with the compiler
      - Before modification

```
;=================================================================
; Initialize standard I/O
;-----------------------------------------------------------------
.if __STANDARD_IO__ == 1
      .glb    _init
      .call   _init,G
      jsr.a   _init
.endif
```

● After modification

```
;==================================================================
; Initialize standard I/O
;------------------------------------------------------------------
.if __STANDARD_IO__ == 1
        .glb   __init
        .call  __init,G
        jsr.a  __init
.endif
```

◆ When you are using the startup file (crt0mr.a30) supplied with the Real Time OS

● Before modification

```
; +------------------------------------------------------+
; | User Initial Routine ( if there are )                |
; +------------------------------------------------------+
; Initialize standard I/O
        .GLB   _init
        JSR.A  _init
```

● After modification

```
; +------------------------------------------------------+
; | User Initial Routine ( if there are )                |
; +------------------------------------------------------+
; Initialize standard I/O
        .GLB   __init
        JSR.A  __init
```

2.2. Bit Widths of Types size_t and ptrdiff_t Changed

Beginning with V.5.40 Release 00, the bit widths of types size_t and ptrdiff_t have both been changed from 16 bits to 32 bits long.

To use types size_t and ptrdiff_t in 16-bit length as will be necessary when you use a user library that uses types size_t and ptrdiff_t that you created with V.5.20 Release 02 or earlier, the following is required.

[1] Select the compile options -fsizet_16 (-fS16) and -fptrdiff_t (-fP16).
[2] Change the standard function library that you use when linking.

Target	Target Library file name to select
M16C/80 /70 Series	nc308_16.lib
M32C/80 Series	nc382_16.lib
M32C/90 Series	nc390_16.lib

● Setup procedure when using the High-performance Embedded Workshop
[1] Select the compile options -fsizet_16 (-fS16) and -fptrdiff_t (-fP16).
[2] From the Build menu of the High-performance Embedded Workshop, select "Renesas M32C Standard Toolchain" and then the C tab.
[3] For Category: on this tab, select "Code Modification."
[4] Select [-fS16 -fP16] from "Miscellaneous options."
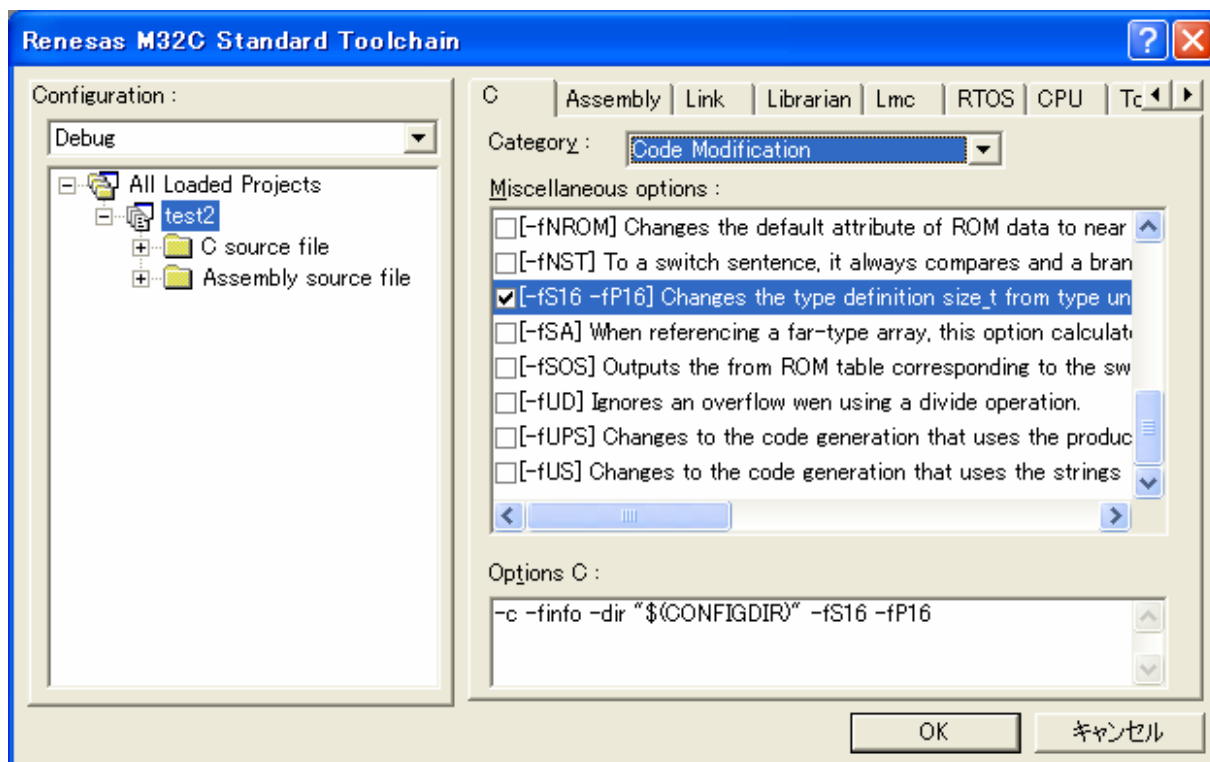
- When using the makefiles generated by the configurator of the real-time OS

  Correct the following part of the makefile.

  The example shown below applies to the M32C/80 series of target microcomputers.

  ◆ Before modification

```
# Use the following macro when you use C-libraries for M32C/80 series.
# NEWLIB = -l nc382lib
```

  ◆ After modification

```
# Use the following macro when you use C-libraries for M32C/80 series.
# NEWLIB = -l nc382_16.lib
```

2.3. Variable Interrupt Vector Table

Beginning with V.5.40 Release 00, if an interrupt function is declared by specifying a vector number, the C compiler will automatically generate a variable interrupt vector table.

In V.5.20 Release 02 or earlier, for variable interrupt vector tables to be automatically generated, you had to select the compile option "-fmake_vector_table (-fMVT)," assemble option "-fMVT" and the link option "-fMVT." You no longer need to select these options.

Note, however, that if the projects created with V.5.20 Release 02 or earlier are converted for use with the new version, these options are not inherited. Therefore, a link error "Can't generate automatically the variable interrupt vector table" will occur when you execute a build process on those projects.

If this error occurs, correct the assembly language startup program "sect308.inc."

```
[sect308.inc: Near the 428th line]
;----------------------------------------------------------------
; variable vector section
;----------------------------------------------------------------
        .section vector,ROMDATA ; variable vector table
        .org VECTOR_ADR
.if     0  ◄───────────────── Inserts .if 0 to disable .lword.
.if     __MVT__ == 0
        .lword  dummy_int       ; BRK (software int 0)
        .lword  dummy_int       ;
        (   )
        .lword  dummy_int       ; software int 63
.endif                          ; __MVT__
.endif  ◄───────────────── Inserts .endif that corresponds to .if 0.
```

2.4. Special Page Vector Table

Beginning with V.5.40 Release 00, if a function to call a special page subroutine is declared by specifying a special page number, the C compiler will automatically generate a special page vector table.
In V.5.20 Release 02 or earlier, for special page vector tables to be automatically generated, you had to select the compile option "-fmake_special_table (-fMST)," assemble option "-fMST" and the link option "-fMST." You no longer need to select these options.

Note, however, that if the projects created with V.5.20 Release 02 or earlier are converted for use with the new version, these options are not inherited. Therefore, a link error "Can't generate automatically the special page vector table." will occur when you execute a build process on those projects.
If this error occurs, correct the assembly language startup program "sect308.inc."

```
[sect308.inc: Near the 500th line]
;================================================================
; fixed vector section
;----------------------------------------------------------------
        .section        svector,ROMDATA      ; specialpage vector table
.if     0  ◄───────────────── Inserts .if 0 to disable SPECIAL.
.if     __MST__ == 0
;================================================================
; special page defination
;----------------------------------------------------------------
;       macro is defined in ncrt0.a30
;       Format: SPECIAL number
;
;----------------------------------------------------------------
;       SPECIAL 255
;       SPECIAL 254
        (   )
;       SPECIAL 19
;       SPECIAL 18
;
.endif ; __MST__
.endif  ◄───────────────── Inserts .endif that corresponds to .if 0.
```

3. C language Startup Program

Startup programs written in C language are supported beginning with the latest version described V.5.40 Release 00. This C language startup program can only be used in combination with V.5.40 Release 00.
This C language startup program can only be used in combination with V.5.40 Release 00.

3.1. File composition of C language Startup Program

C language Startup Program contains the following 13 files.
(1) resetprg.c
   Initializes the microcomputer.
(2) initsct.c
   Initializes each section (by clearing them to 0 and transferring initial values).
(3) heap.c
   Reserves storage for the heap area.
(4) fvector.c
   Defines the fixed vector table.
(5) intprg.c
   Declares the entry function for variable vector interrupts.
(6) firm.c / firm_ram.c
   Reserves storage for the program and workspace areas used by firm of FoUSB/NSD as dummy areas when OnChipDebugger is selected.
(7) cregdef.h
   Declares the internal registers of the microcomputer.
   Please do not alter the file.
(8) heapdef.h
   Initializes the heap area.
(9) initsct.h
   Contains statements for the processes (assembler macros) that initialize each section.
   Please do not alter the file.
(10) resetprg.h
   Include does each header file for C language Startup Program.
(11) stackdef.h
   Defines the stack size.
(12) vector.h
   Defines the variable vector address.
(13) typedef.h
   Declares each type by typedef.

3.2. Processing of C language startup program

3.2.1. resetprg.c

The content of this file varies with the selected MCU.
This file is necessary for C language Startup Program.

```
#include "resetprg.h"
//////////////////////////////////////
// declare sfr register
#pragma ADDRESS  protect  0AH
#pragma ADDRESS  pmode0   04H
#pragma ADDRESS  _SB__    0400H
_UBYTE protect,pmode0;
_UBYTE _SB__;


#pragma entry start
void start(void);
extern void initsct(void);
extern void _init(void);
void exit(int);

#pragma section program  interrupt                              → (1)
#pragma inline set_cpu()
void set_cpu(void)                                              → (2)
{
        _isp_   = &_istack_top;  // set interrupt stack pointer → (3)
        protect = 0x02;          // change protect mode register → (4)
        pmode0  = 0x00;          // set processor mode register → (5)
        protect = 0x00;          // change protect mode register → (6)
        _flg_   = 0x0080;               // set flag register     → (7)
        _sp_    = &_stack_top;   // set user stack pointer       → (8)
        _sb_    = (char _far *)0x400;    // 400H fixation (Do not change) → (9)
        _asm("  fset    b");
        _sb_    = (char _far *)0x400;
        _asm("  fclr    b");
        _intb_ = (char _far *)VECTOR_ADR; // set variable vector's address → (10)


}


void start(void)
{
        set_cpu();                       // initialize mcu       → (11)
        initsct();                       // initlalize each sections → (12)
#ifdef __HEAP__
        heap_init();            // initialize heap               → (13)
#endif
#ifdef __STANDARD_IO__
        _init();                // initialize standard I/O       → (14)
#endif
        _fb_ = 0;                        // initialize FB registe for debugger
        main();                 // call main routine             → (15)


        exit(0);                // infinite loop
}


void exit(int rc)
{
        while(1);
}
```

(1) The startup function is located in the interrupt section.

(2) Declares the function body of the CPU initialization function set_cpu().

(3) Initializes the interrupt stack pointer.

(4) Write enables the protect mode register.

(5) Sets the processor mode register to "single-chip mode". If modes need to be changed, this expression must be altered.

(6) Write protects the protect mode register.

(7) Sets the U flag to 1 (stack pointer changed for the user stack).

(8) Initializes the user stack pointer.

(9) Sets the SB register to address 0x400 (which sets the start address of RAM).

(10) Sets the variable vector address in the INTB register. The VECTOR_ADR that defines the variable vector address is defined in vector.h. Note also that if the variable vector address is altered by a link option for section order under the HEW environment, resetprg.c must always be recompiled.

(11) Calls the CPU initialization function.

(12) Initializes each section (by clearing them to 0 and transferring initial values).

(13) Initializes the heap area.If memory management functions are used, call to this function must be enabled.

(14) Initializes the standard input/output device.If standard input/output functions are used, call to this function must be enabled.

(15) Calls the main function.

3.2.2. resetprg.h

Include does each header file for C language Startup Program.
This file is necessary for C language Startup Program.

3.2.3. initsct.c

The content of this file varies with the selected MCU.
This file is necessary for C language Startup Program.

```c
#include "initsct.h"
void initsct(void);

void initsct(void)
{
        sclear("bss_SE","data,align");                          → (1)
        sclear("bss_SO","data,noalign");
        sclear("bss_NE","data,align");
        sclear("bss_NO","data,noalign");
        sclear("bss_FE","data,align");                          → (2)
        sclear("bss_FO","data,noalign");

        /* clear bss for NSD */
        sclear("bss_MON1_E","data,align");
        sclear("bss_MON2_E","data,align");
        sclear("bss_MON3_E","data,align");
        sclear("bss_MON4_E","data,align");
        sclear("bss_MON1_O","data,noalign");
        sclear("bss_MON2_O","data,noalign");
        sclear("bss_MON3_O","data,noalign");
        sclear("bss_MON4_O","data,noalign");

        // when add new sections
        // bss_clear("new section's name");
```

```
        scopy("data_SE","data,align");                              → (3)
        scopy("data_SO","data,noalign");
        scopy("data_NE","data,align");
        scopy("data_NO","data,noalign");


        /* copy data section for NSD */
        scopy("data_MON1_E","data,align");
        scopy("data_MON2_E","data,align");
        scopy("data_MON3_E","data,align");
        scopy("data_MON4_E","data,align");
        scopy("data_MON1_O","data,noalign");
        scopy("data_MON2_O","data,noalign");
        scopy("data_MON3_O","data,noalign");
        scopy("data_MON4_O","data,noalign");
        scopy("data_FE","data,align");                               → (4)
        scopy("data_FO","data,noalign");
}
```

(1) sclear: Clears the bss section of the near area to zero.

If the bss section name is altered or a new bss section name is added using the #pragma SECTION bss feature, NE and NO must be altered or added in pairs.

```
        sclear( "section-name_NE" , "data.align" );
        sclear( "section-name_NO" , "data,noalign" );
```

Example: When a section is added by #pragma section bss bss2, the following must be added to init.sct.c

```
        sclear( "bss2_NE" , "data.align" );
        sclear( "bss2_NO" , "data,noalign" );
```

(2) sclear_f: Clears the bss section of the far area to zero.

If an external variable without initial values is declared using the far qualifier, this macro function must be enabled.

(3) scopy: Transfers initial values to the data section of the near area.

If the data section name is altered or a new dada section name is added using the #pragma SECTION data feature, NE and NO must be altered or added in pairs.

```
        sclear( "section-name_NE" , "data.align" );
        sclear( "section-name_NO" , "data,noalign" );
```

Example: When a section is added by #pragma section data data2, the following must be added to initsct.c

```
        sclear( "data2_NE" , "data.align" );
        sclear( "data2_NO" , "data,noalign" );
```

(4) scopy_f: Transfers initial values to the data section of the far area.

If an external variable with initial values is declared using the far qualifier, this macro function must be enabled.

3.2.4. initsct.h

This file is necessary for C language Startup Program.
Please do not alter the file.

3.2.5. heap.c

Only when memory management functions such as malloc are used.

```
#include "typedefine.h"
#include "heapdef.h"
#pragma SECTION  bss      heap                                      → (1)


_UBYTE heap_area[__HEAPSIZE__];                                     → (2)
```

(1) Locates the heap area in the heap_NE section.

If the heap size consists of an odd number of bytes, the heap_NO section is assumed by default.

(2) Reserves storage for the heap area by an amount equal to the size defined in __HEAPSIZE__.

3.2.6. heapdef.h

This file is necessary for memory management functions.

```
extern   _UBYTE _far * _mnext;
extern   _UDWORD _msize;
///////////////////////////////////////
// It's size of heap
// When you want to change size of heap,
// please change this line.
// When you change this line,
// you must modify the value using hex character.


#ifndef __HEAPSIZE__
#define __HEAPSIZE__      0x300
#endif
extern _UBYTE heap_area[__HEAPSIZE__];


#pragma inline heap_init()
void heap_init(void)
{
        _mnext = &heap_area[0];                                    → (1)
        _msize = __HEAPSIZE__;                                     → (2)
}
```

(1) Initializes the heap management area.

(2) Initializes the heap size.

3.2.7. fvector.c

This file is necessary for C language Startup Program.

```
#include "vector.h"
#pragma sectaddress        fvector,ROMDATA Fvectaddr                → (1)


//////////////////////////////////////////////////////////////////////

#pragma interrupt/v _dummy_int    //udi                            → (2)
#pragma interrupt/v _dummy_int    //over_flow
#pragma interrupt/v _dummy_int    //brki
#pragma interrupt/v _dummy_int    //address_match
#pragma interrupt/v _dummy_int    //single_step
#pragma interrupt/v _dummy_int    //wdt
#pragma interrupt/v _dummy_int    //dbc
#pragma interrupt/v _dummy_int    //nmi
#pragma interrupt/v start                                          → (3)



#pragma interrupt _dummy_int()
void _dummy_int(void){}
```

(1) Outputs the section and address of a fixed vector table.

This pragma is used exclusively for startup and cannot normally be used.

(2) Fills fixed vectors other than reset with a dummy function (_dummy_int).

This pragma is used exclusively for startup and cannot normally be used.

(3) Defines the entry function.

The function to be executed upon reset is registered in a fixed vector.

3.2.8. intprg.c

The content of this file depends on the MCU.

```
// BRK (software int 0)
#pragma interrupt        _brk(vect=0)
void _brk(void){}


// vector 1 reserved
// vector 2 reserved
// vector 3 reserved
// vector 4 reserved
// vector 5 reserved
// vector 6 reserved


// A/D1 (software int 7)
#pragma interrupt        _ad1(vect=7)
void _ad1(void){}


// DMA0 (software int 8)
#pragma interrupt        _dma0(vect=8)                                    → (1)
void _dma0(void){}


// DMA1 (software int 9)
#pragma interrupt        _dma1(vect=9)
void _dma1(void){}


// DMA2 (software int 10)
#pragma interrupt        _dma2(vect=10)
void _dma2(void){}


// DMA3 (software int 11)
#pragma interrupt        _dma3(vect=11)
void _dma3(void){}


// TIMER A0 (software int 12)
#pragma interrupt        _timer_a0(vect=12)
void _timer_a0(void){}


// TIMER A1 (software int 13)
#pragma interrupt        _timer_a1(vect=13)
void _timer_a1(void){}


        :
      (omitted)
        :
```

(1) Declares the variable vector interrupt function.

The functions corresponding to each variable vector interrupt function are declared. A variable vector table is generated at the same time.

3.2.9. firm.c / firm_ram.c

The content of this file is altered depending on the microcomputer type and selected OnChipDebugger.

```
#include "typedefine.h"
#pragma section bss FirmRam                                      → (1)
_UBYTE _workram[0x4];     // for Firmware's workram            → (2)
```

(1) Allocates the work ram area to be used by the E8 firmware in the FirmRam_NE section.
(2) Reserves storage for the work ram area by an amount equal to the size defined in __WORK_RAM__.

3.2.10. cregdef.h

This file is necessary for C language Startup Program.
Please do not alter the file.

3.2.11. stackdef.h

This file is necessary for C language Startup Program.

```
#ifndef __STACKSIZE__
#pragma STACKSIZE          0x300                                 → (1)
#else
#pragma STACKSIZE          __STACKSIZE__                         → (2)
#endif
#ifndef ISTACKSIZE
#pragma ISTACKSIZE         0x300                                 → (3)
#else
#pragma ISTACKSIZE         __ISTACKSIZE__                        → (4)
#endif
extern _UINT _stack_top,_istack_top;
```

(1) Indicates the default size of the user stack.
(2) Outputs a user stack section and reserves storage for it.
(3) Indicates the default size of the interrupt stack.
(4) Outputs an interrupt stack section and reserves storage for it.

3.2.12. vector.h

This file is necessary for C language Startup Program.

```
#define Fvectaddr       0xffffdc                                 → (1)
#ifndef VECTOR_ADR
#define VECTOR_ADR      0x0fffd00                                → (2)
#endif
```

(1) Indicates the start address of a fixed vector table.
(2) Indicates the start address of a variable vector table.
     If the start address of a variable vector table is changed, the address that is set in the INTB register in resetprg.c must also be changed at the same time.

3.2.13. typedefine.h

This file is necessary for C language Startup Program.
Please do not alter the file.

3.3. C language Startup Program is used on High-performance Embedded Workshop.

(1) "Csource startup Application" of   "New Project Workspace" is selected, and Workspace is made.
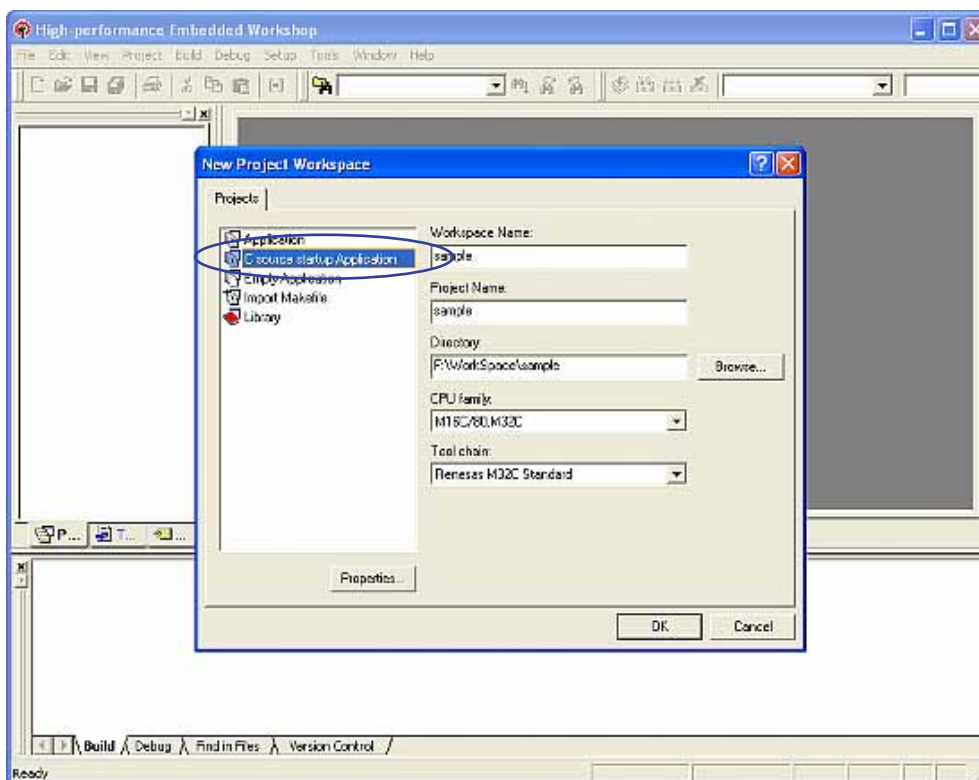
If while multiple compilers are installed in your computer you select another microcomputer for the CPU type after selecting C source startup Application, the focus for C source startup Application will move to Application, with the result that the selected C source startup has no effect. In such a case, therefore, select "C source startup Application" again.

(2) The target microcomputer is selected from "CPU Series" and "CPU Group".
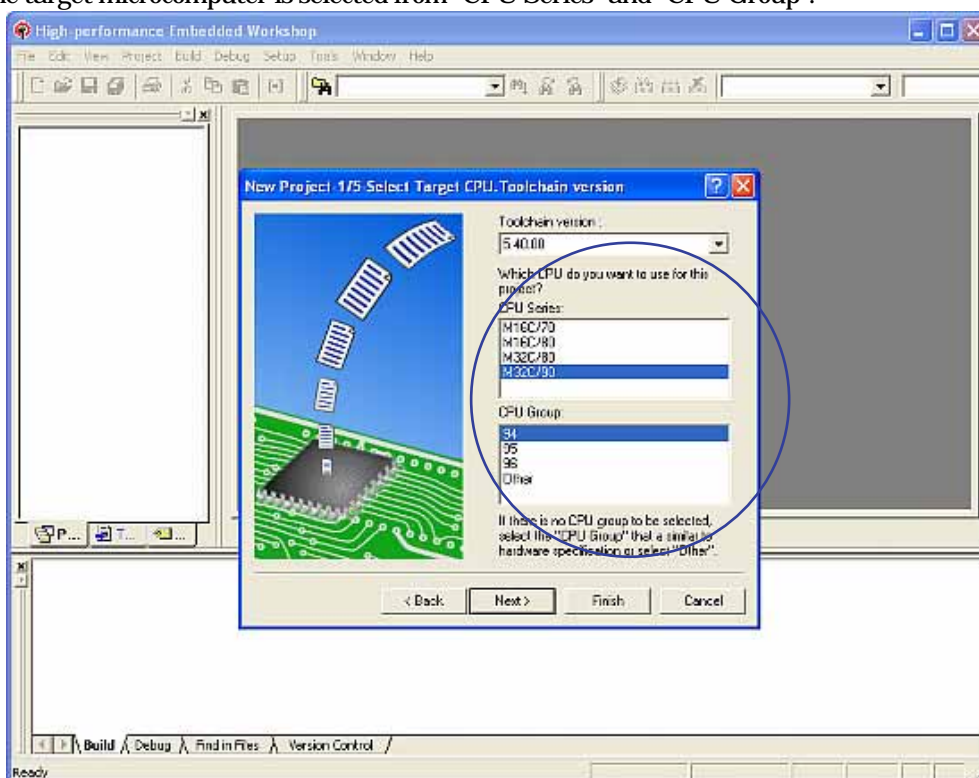
When a type of microcomputer is selected, its corresponding sfr header file is copied to the workspace. Furthermore, a variable vector table (intprg.c) is registered.

(3) Settings for the case where the standard function and memory management function libraries are used
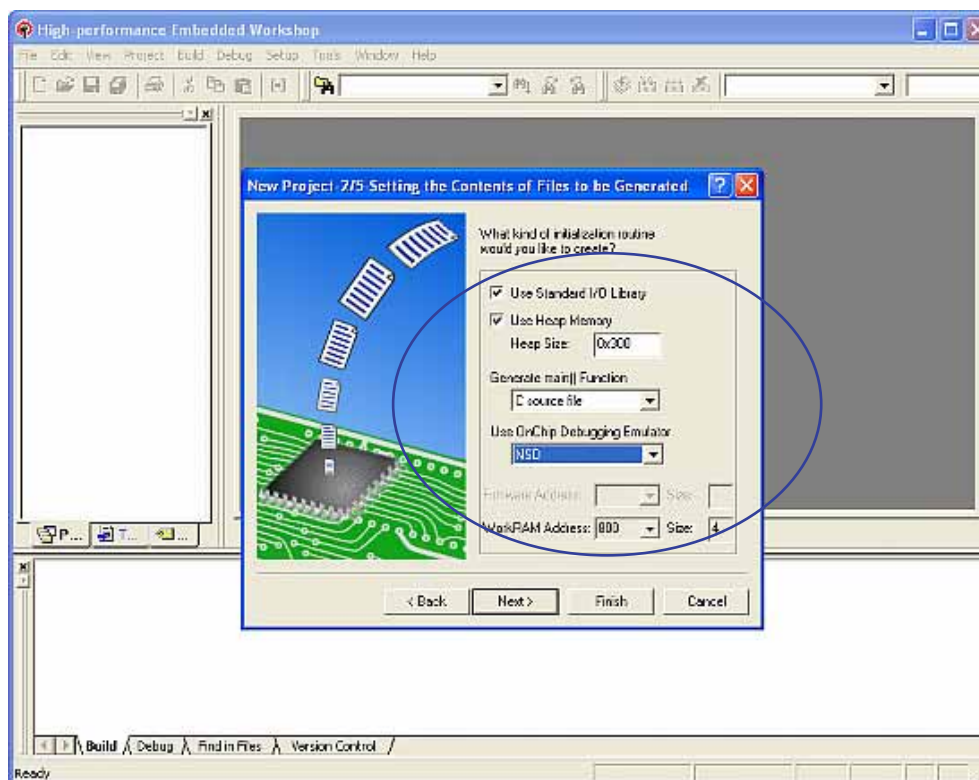
[1] Select this check box when you use the standard function library.

When this check box is selected (flagged with a check mark), function calls to _init() in resetprg.c are enabled.

Furthermore, device.c and init.c are registered to the project.

[2] Select this check box when you use the memory management function library.

When this check box is selected (flagged with a check mark), function calls to heap_init() in resetprg.c are enabled.

Furthermore, heapdef.h and heap.c are registered to the project.

[3] Select the appropriate debugger when you use OnChip Debugging Emulator.
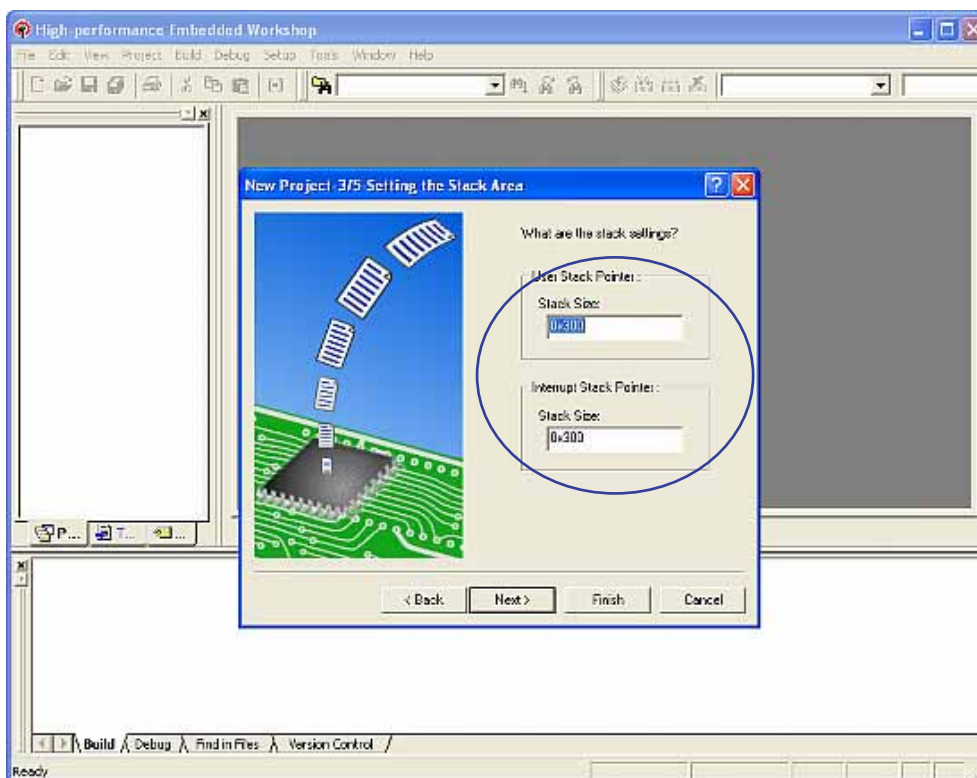
The selectable debuggers are FoUSB and NSD.

Note, however, that you cannot select either one of the two or both depending on the selected type of microcomputer.

When this selection is made, firm.c is registered.

(4) Selecting the stack size

[1] Set the user stack size.

When this stack size is set, stackdef.h is registered.

[2] Set the interrupt stack size.

When this stack size is set, stackdef.h is registered.

To change the stack and HEAP sizes after creating a project, alter the value of each of the following in compile option settings:

```
#define Fvectaddr        0xffffdc                                    → (1)
#ifndef VECTOR_ADR
#define VECTOR_ADR       0x0fffd00                                   → (2)
#endif
```

(5) List of registered files
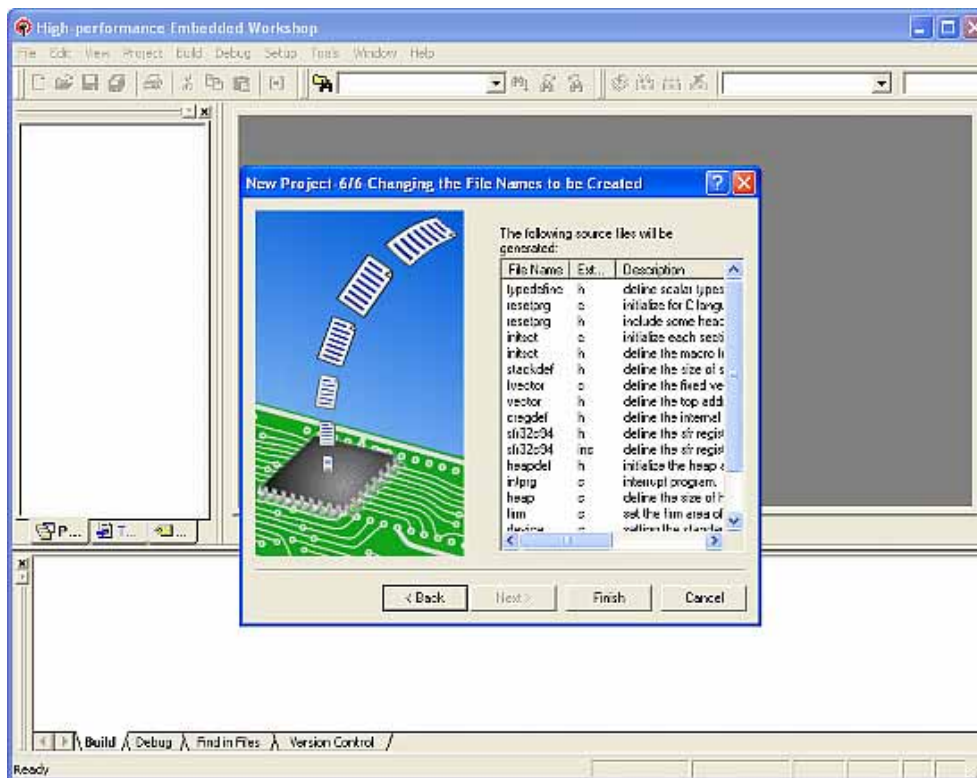
Here, you can check the list of files to be registered.

However, since the SFR header (C language header or assembler header) registered for each type of microcomputer is only copied to the workspace, take a look at this list to confirm the file name.

(6) Section Order

To confirm the order in which sections are linked and the addresses to which they are linked, take a look at "Category": Section Order in "Link" of "Renesas M32C Standard Toolchain".
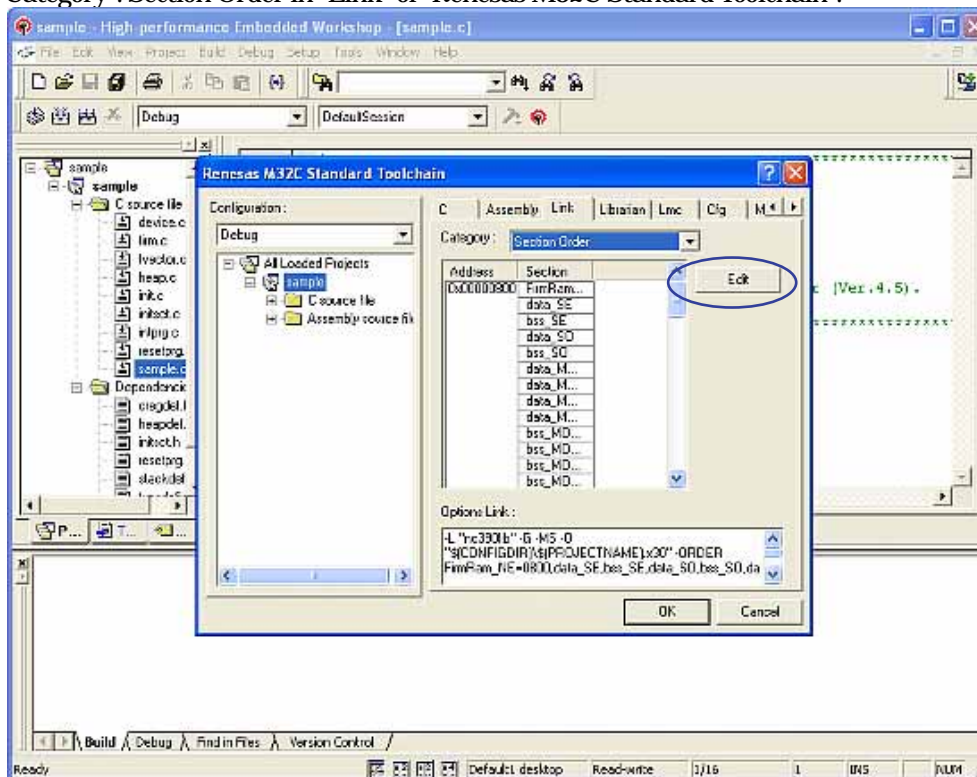
If you added a new section with #pragma SECTION, click the [Edit] button in (1) to open the Section window.

3.4. Assembly language Startup Program is used on High-performance Embedded Workshop.

"Application " of "New Project Workspace" is selected, and Workspace is made.

4. A Guide to Porting Projects Created with TM to High-performance Embedded Workshop V.4

This document explains how to port projects created with TM V.2.xx or V.3.xx into High-performance Embedded Workshop V.4.
Note, however, that due to upgrading of the High-performance Embedded Workshop or for other reasons, the guide described here may not always apply. For the latest information, please refer to the FAQ section of the Renesas Development Environment Web site.

4.1. Summary

To port projects created using TM V.2.xx or V.3.xx into High-performance Embedded Workshop V.4, the Import Makefile function of High-performance Embedded Workshop is used. This function can create projects from such items of information as source files and build options described in the specified makefile files.
In TM, project files are created in the makefile format executable in GNU make format. When project files created with TM are selected as makefile files using High-performance Embedded Workshop Import Makefile function, they are converted to files that can run in High-performance Embedded Workshop. In addition to TM project files, the Import Makefile function can also convert files in the makefile formats for hmake, nmake, and gmake to High-performance Embedded Workshop projects.

4.2. Porting Procedure

To port projects created using TM into High-performance Embedded Workshop, perform the following steps:
 (1) Open the File menu and select the New Workspace command.
 (2) The New Project Workspace dialog box opens.
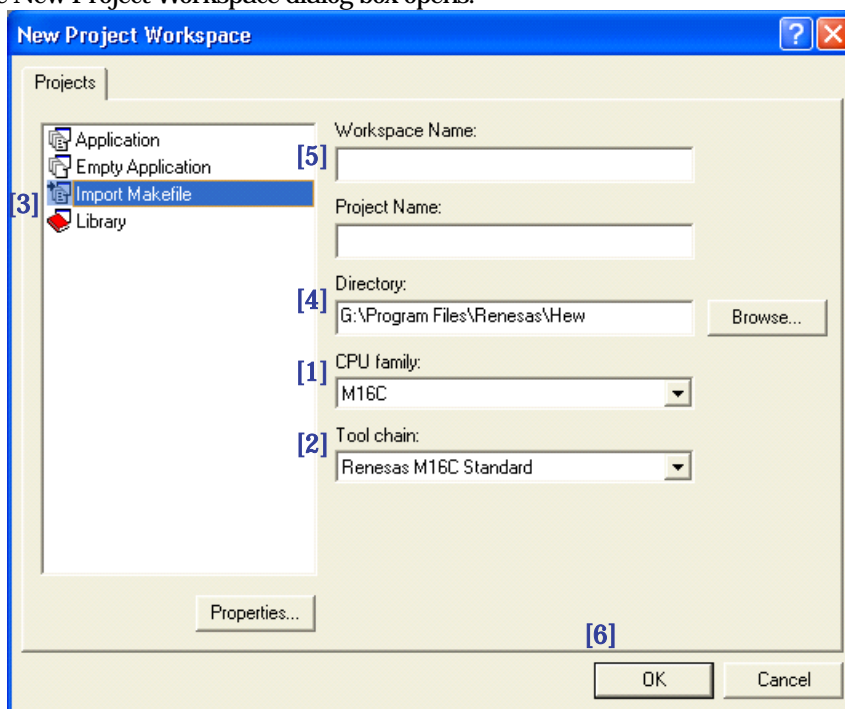
[1] Select the type of CPU used in the TM project from the Type of CPU drop-down list.

[2] Select the tool chain (cross tool) used for the TM project from the Toolchain drop-down list. The names of tool chains and corresponding cross tools are shown in Table 1.

Tool Chain	Cross Tool
Renesas M16C Standard	NC30WA
Renesas R8C Standard	NC8C
Renesas M32C Standard	NC308WA
Renesas M32R Standard	CC32R

[3] Select Import Makefile from the Project list.

[4] Type the directory path in the Directory text box.

[5] Type the workspace name in the Workspace Name text box. The same name will be automatically entered as the project name in the Project Name text box.

[6] Click OK.

(3) You should now be able to see the New Project-1/4-Import Makefile wizard.
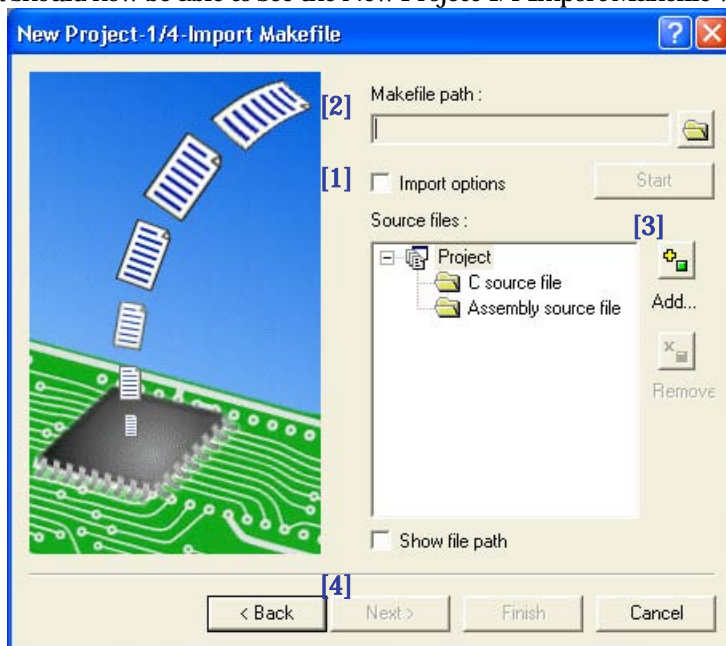
[1] Select the Import options check box; this will enable information on build options (compiling and assembling options etc.) to be used to create High-performance Embedded Workshop projects. If you clear the Import options check box, the above information is neglected and not used in High-performance Embedded Workshop.

[2] Type the name of the TM project file (with extension .tmk) in the Makefile path text box. As soon the name is input, the specified file is analyzed, and upon analysis completion, the analyzed source files are displayed in a tree structure in the Source files box. Click the Start button to analyze the specified file again.

[3] If there are any errors in the analysis results (tree structure in the Source files box), rectify the tree structure with the Add and Remove buttons.

[4] Click Next.

(4) Follow the instructions according to the Wizard as it continues in the procedure.

4.3. Usage Notices

4.3.1. TM-to-High-performance Embedded Workshop Portable and Non-Portable Information

When you port a project created using TM into High-performance Embedded Workshop, not all the components of the project can be ported.
Portable information is as follows:
◆ Paths of assembler source files
◆ Paths of C-language source files
◆ Assembling options
◆ C-compiling options
◆ Linking options (except linkage order)
Non-Portable Information:
◆ Linkage order
◆ Tool configurations, dependencies, and options other than Assembler, C Compiler,Linker
To transfer these items, edit the High-performance Embedded Workshop project as described in Section 3.4 and further after processing the Import Makefile.

4.3.2. Cross Tools

Import Makefile cannot enable all cross tool versions for use in High-performance Embedded Workshop projects regardless of whether they are used with TM or not; only the following cross tools versions are valid for High-performance Embedded Workshop projects:
NC30WA   :   V.5.20 Release1 or later

NC8C	:	V.5.30 Release1
NC308WA	:	V.5.20 Release1 or later
CC32R	:	V.4.20 Release1 or later

4.3.3. High-performance Embedded Workshop Versions

When TM projects are ported into High-performance Embedded Workshop, information portable to High-performance Embedded Workshop varies according to the High-performance Embedded Workshop version. The information that can be ported from each cross tool to various High-performance Embedded Workshop versions are shown.

| | | High-performance Embedded Workshop | |
		V.3	V.4
NC30WA	V.5.20 Release1	C	C
	V.5.30 Release1	C	C
	V.5.30 Release 02	---	A
	V.5.40 Release00	---	A
NC8C	V.5.30 Release1	C	C
NC308WA	V.5.20 Release1	C	C
	V.5.20 Release 02	---	B
	V.5.40 Release00	---	A
CC32R	V.4.20 Release1	C	C
	V.4.20 Release1A	C	C
	V.4.30 Release 00	B	B
	V.5.00 Release 00	---	A

A : All the items of information listed in Section 3.1 are portable.
B : The compiler and assembler of option and the paths of assembler and C-language source files are portable.
C : Only the paths of assembler and C-language source files are portable.

4.3.4. Generated Project Workspace

Because the project workspace created for a TM project ported to the High-performance Embedded Workshop environment is simply the contents of the makefile itself, its configuration (object output directory) will be different than that of a newly generated project workspace in High-performance Embedded Workshop.
To validate the configuration, modify the output directory file names for the compiler, assembler and linker as follows:

| Output Directory (compiler, assembler) | : | $(CONFIGDIR) |
| Output Directory (linker) | : | $(CONFIGDIR)¥$(PROJECTNAME).x30 |

4.3.5. Load Module Converter

Import Makefile cannot port the information contained in any load module converter (for example, information on options, command executions, or dependencies) into the High-performance Embedded Workshop project. If using a load module converter to create projects in TM, change the settings of the load module converter as follows after completing the Makefile processing:
 (1) Open the Build menu and select the Build Phases command.
 (2) The Build Phases dialog box will open.
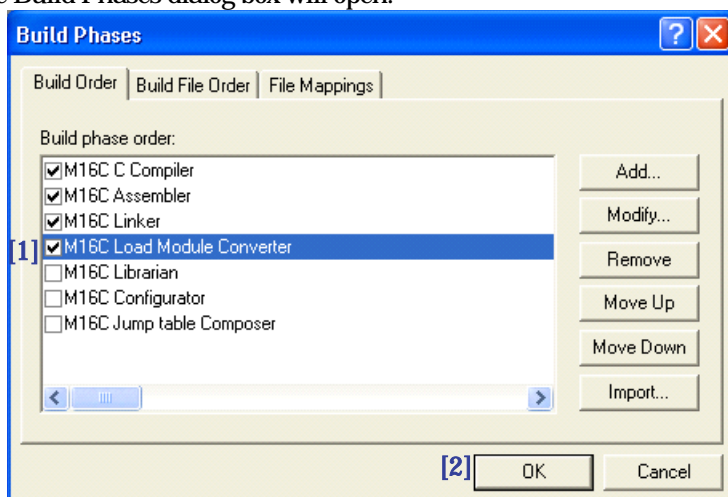
[1] Select the Mxxx Load Module Converter check box from the Order of Build Phases list.

[2] Click OK.

(3) Open the Build menu and select Renesas Mxxx Standard Toolchain.

(4) The Renesas Mxxx Standard Toolchain dialog box appears.
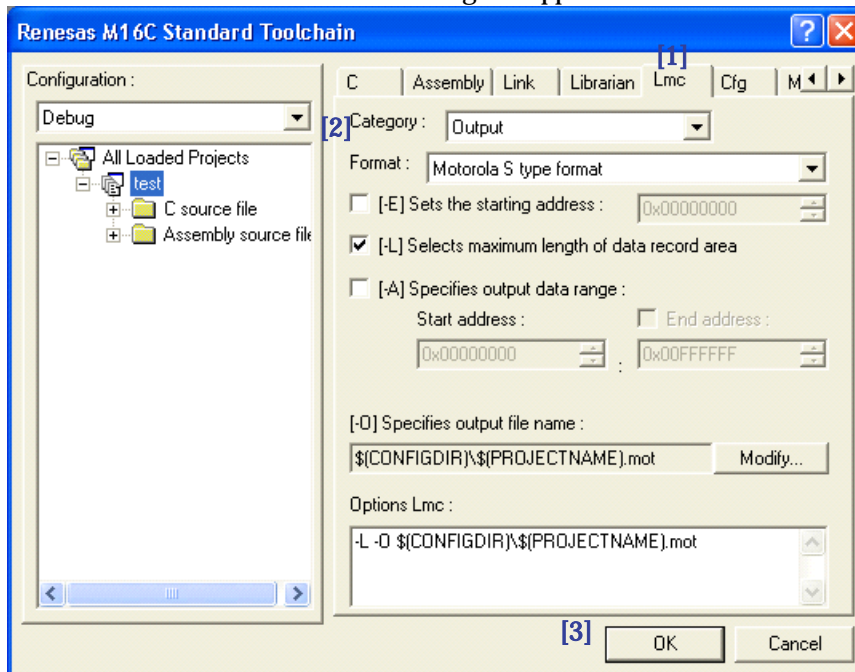
[1] Click the Lmc tab.

[2] Select the Category type from the Category drop-down list.

[3] Click OK.

4.3.6. Other Tools

Import Makefile cannot port any information (options, command executions, dependencies) contained in tools other than the assembler, C compiler, and linker. If any tools other than the assembler, C compiler, linker, and load module converter are used to create projects in TM, custom build phases must be created in High-performance Embedded Workshop. Custom build phases are specifically for operating other tools before, after, or during standard builds (in the assembler, C compiler, and linker).

For more details, see Section 3.2 "Creating Custom Build Phases" in the High-performance Embedded Workshop 3 User's Manual. The following is provided as an example of how to register the cross-reference generation tool xrf30 with High-performance Embedded Workshop.

(1) Open the Build menu and select the Build Phases command.
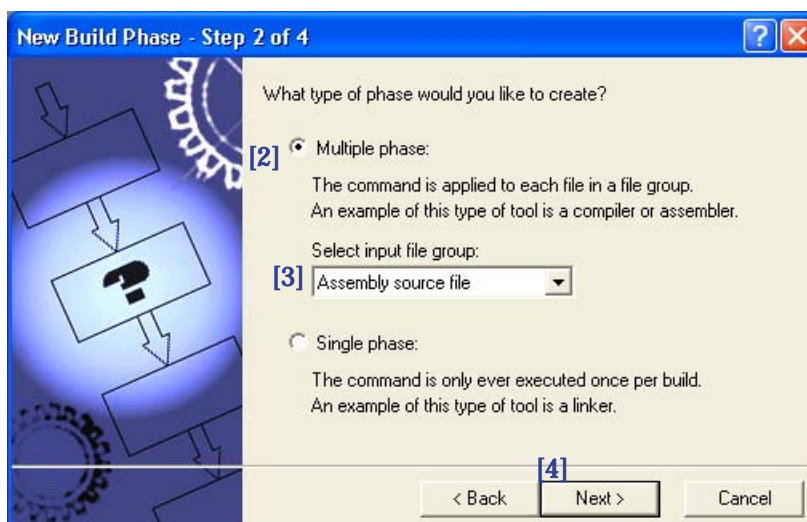
(2) The Build Phases dialog box appears.
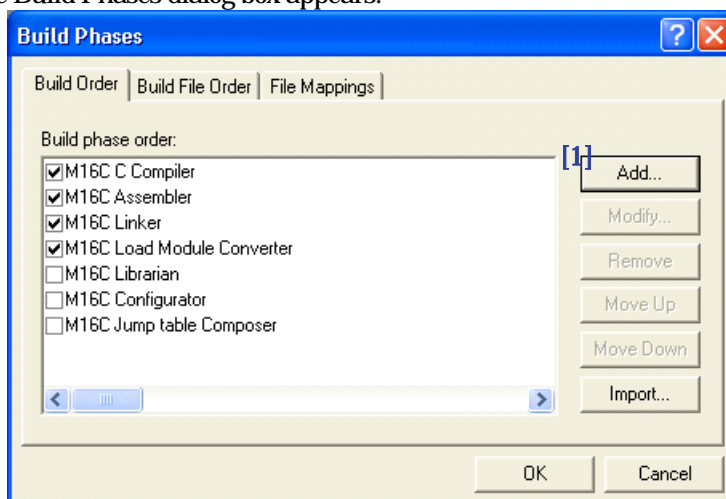
[1] click Add.

(3) The New Build Phase- Step 1/4 wizard opens. Follow the instructions to register the tool as follows:
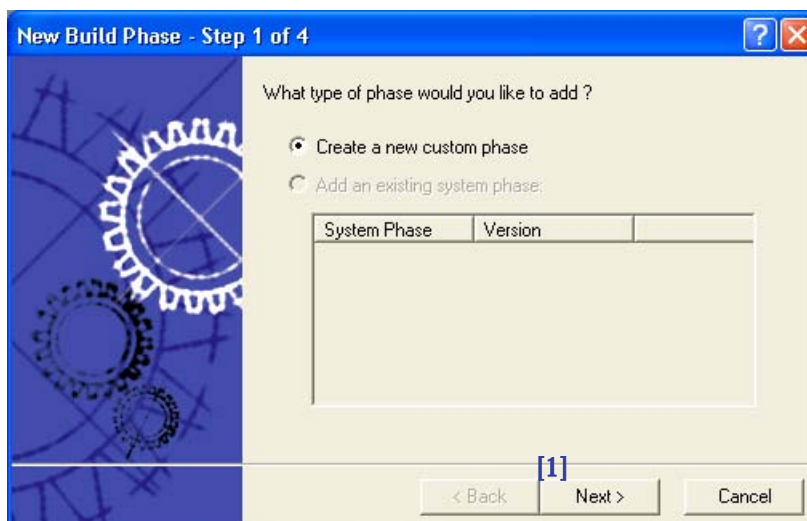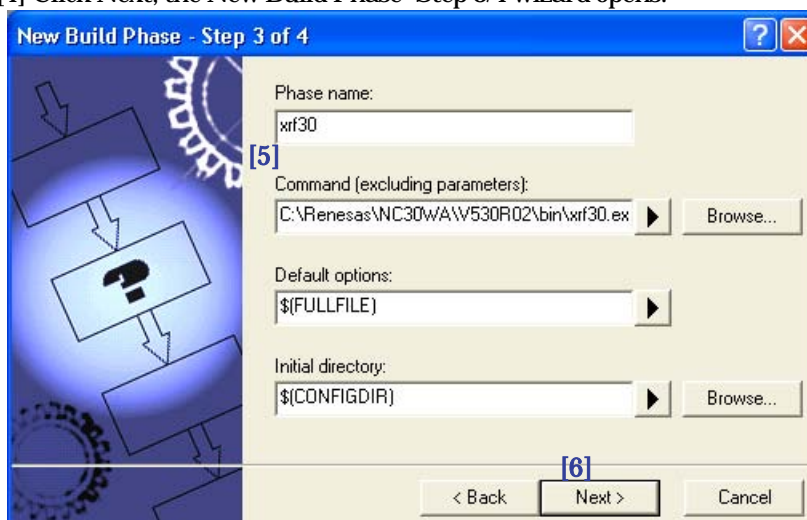
[1] Click Next (the Create a New Custom Phase check box is selected by default); the New Build Phase-2/4 Step wizard opens.

[2] In this wizard, select the Multiple Phase check box.
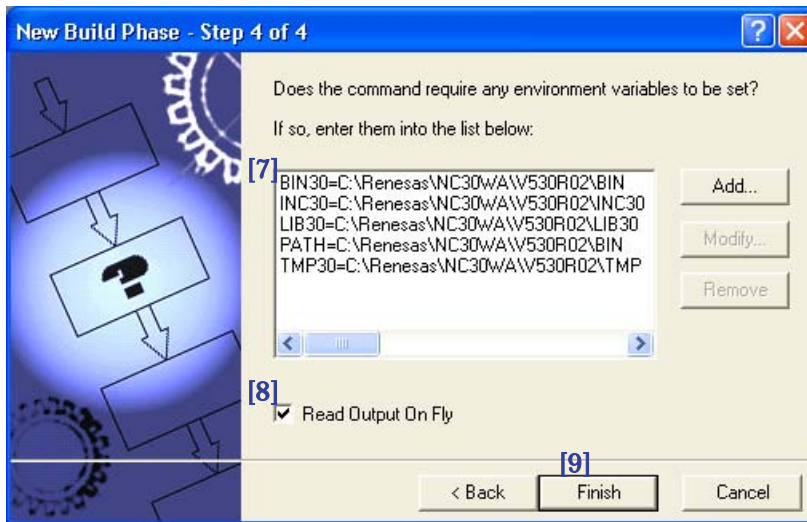[3] Select Assembly Source file from the Select input file group.
[4] Click Next; the New Build Phase- Step 3/4 wizard opens.

[5] Type xrf30 and its fullpath name in the Phase Name and the Command text box.
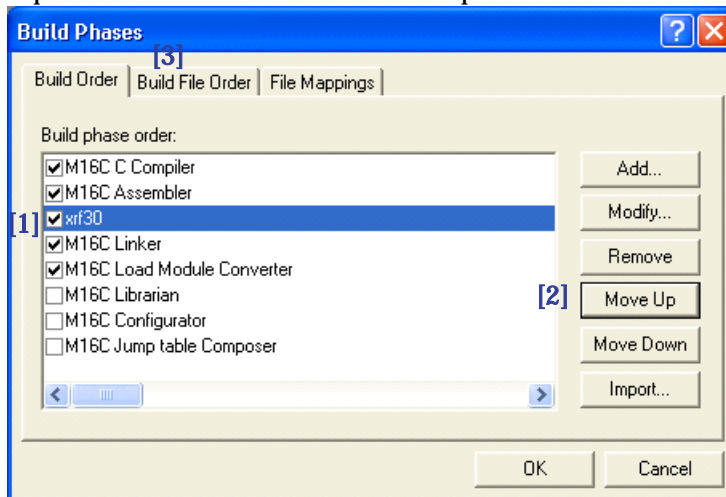[6] Click Next; the New Build Phase- Step 4/4 wizard opens.

[7] In this wizard, enter the necessary environment variables in the list.

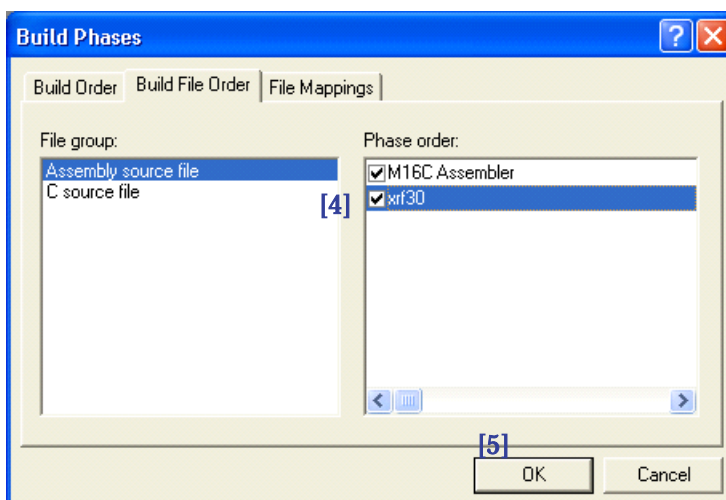[8] Select "Read Output On Fly" check box.

[9] Click Finish.

(4) You return to the Build Phases dialog box at this point, where you can see that xrf30 has been registered as a build phase at the end of the Order of Build phase order.

[1] Select xrf30 from the Order of Build phase order.

[2] Click Move Up to move xrf30 next to the assembler name (see Figure 10).

[3] Click the Build File Order tab.

[4] Select the xrf30 check box in the Order of Phase order.

[5] Click OK.

(5) Open the Options menu and select the xrf30 command.

(6) The xrf30 Options dialog box appears; select options as necessary. This setting executes xrf30 for all assembler source files after assemble is completed at a build (before linking files).

4.3.7.　Linkage order

Import Makefile cannot port the linking order information to High-performance Embedded Workshop. High-performance Embedded Workshop arranges the linking order alphabetically. To change this order, go through the following steps:

(1) Open the Build menu and select the Linkage Order command.

(2) The Linkage Order dialog box opens.
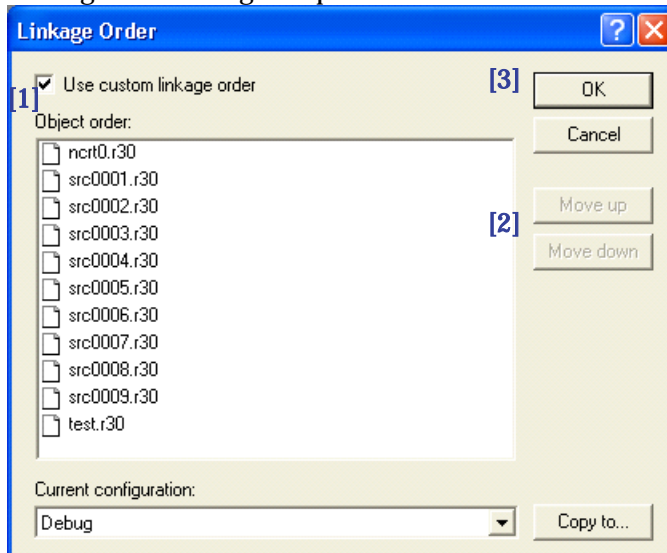
[1] Select "Use custom linkage order" check box.

[2] Select a file from the Object order list, and click Move up or Move down to move the file. Repeat this step for all files that need to be rearranged.

[3] Click OK.

4.3.8.　Placing the Start Up program at the top of Linkage Order

As the Import Makefile cannot port linking order information to High-performance Embedded Workshop, and links are order alphabetically, the start up program may not be placed at the top of the linking order. To place it at the top, follow the steps described previously in "4.3.7. Linkage order".

5.   Function added from V.5.41 Release 00

5.1.   Support for Call Walker

Beginning with V.5.41 Release 00, in addition to the STK viewer, Call Walker is available to use. It permits you to calculate the stack size used in an application program.
For details on how to use Call Walker, click Help Topics on the Help menu of Call Walker and read the ensuing Call Walker Help.

5.1.1.   Starting Call Walker

- Startup
    You can start Call Walker using one of the following two methods.
    [1] To start Call Walker from the High-performance Embedded Workshop
        Click Renesas Call Walker on the Tool menu of the High-performance Embedded Workshop.
    [2] To start Call Walker from Windows Start menu
        In All Programs4 of Windows Start menu, locate the Renesas menu labeled "M32C-90,80, M16C-80,70 Series C Compiler V.5.41 Release 01" and then click Call Walker in it.
- Termination
    Click Exit on the File menu of Call Walker.

5.1.2.   Creating Input Files for Call Walker

Use the .sni file creation tool named gensni to create the input files for Call Walker.
The method for creating the input files for Call Walker differs depending on how the absolute module files (x30) are built.
 (1) When built in the High-performance Embedded Workshop
    When you build an x30 file, gensni is automatically executed.
 (2) When compiled, assembled and linked at the command prompt (or DOS prompt)
    Execute gensni at the command prompt (or DOS prompt).

```
[Example for executing gensni]


       C:¥> gensni  -o  sample.sni  sample.x30
```

5.1.3.   Selecting an Input File for Call Walker

To select an input file for Call Walker, click Import Stack File on the File menu of Call Walker and then select one in the Stack File window that is displayed.

5.2.   Support for the Map Section Information Window of the High-performance Embedded Workshop

Beginning with V.5.41 Release 00, in addition to the MAP viewer, the Map Section Information window of the High-performance Embedded Workshop is available to use. This window permits you to display the map information of absolute module files.
For details on how to use the Map Section Information window, refer to Section 13, "Map," of the High-performance Embedded Workshop V.4.01 User's Manual.