CUSTOMER NOTIFICATION

SUD-DT-04-0154

March 26, 2004

Koji Nishibayashi, Senior System Integrator
Microcomputer Group
2nd Solutions Division
Solutions Operations Unit
NEC Electronics Corporation

CP(K), O

V850 Series C Compiler Package

CA850 Ver. 2.70

Operating Precautions

Be sure to read this document before using the product.

CONTENTS

This document explains the modifications from V850 Series C compiler package CA850 Ver. 2.61 to Ver. 2.70, cautions, and usage restrictions. Be sure to read this document before using CA850 Ver. 2.60.

1 USER'S MANUAL

The following user's manuals are available for this version as a PDF file only.

Document Name	Document Number
CA850 C Compiler Package - C Language	U16930E
CA850 C Compiler Package - Assembly Language	U16931E
CA850 C Compiler Package - Operation	U16932E
CA850 C Compiler Package - Coding Technique	U16076E
PM plus Ver. 5.20	U16934E

Only the Windows version PM plus user's manual is supplied.
The "CA850 C Compiler Package - Coding Technique" targets Ver. 2.50, but it can also be used for the manual of Ver. 2.70.

2 SUPPORTED TOOLS

Use the following version of PM plus when using CA850 Ver. 2.61.

- PM plus V5.20

Note that the following version of the project manager is not supported.

- PM plus Ver. 5.10 or earlier
- All versions of PM

Use the following versions of the integrated debugger or system simulator when using in combination with CA850 Ver. 2.70 and PM plus.

Debugger/Simulator Name	Version
Integrated debugger ID850	Ver. 2.50 or later
Integrated debugger ID850NW	Ver. 2.50 or later
Integrated debugger ID850NWC	Ver. 2.50 or later
System simulator SM850	Ver. 2.50 or later

3 CHANGES FROM Ver. 2.61 TO Ver. 2.70

This section explains changes from Ver. 2.61 to Ver. 2.70.

3.1 Changes in C Compiler

3.1.1 Change of -cn and -cnv850e options

The “-cn” option, which is used to specify compilation as a V850 object, and the “-cnv850e” option, which is used to specify compilation as a V850E object, have been added. (These options are provided in the assembler (as850) and they have now been added to the compiler in Ver. 2.70.)
Device file specification was essential when creating an object common to V850 and V850E in Ver. 2.61 or earlier, but it is no longer required in Ver. 2.70.
If neither the -cn/-cnv850e option nor device file is specified, compilation is stopped.
An error is output if the -cn or -cnv850e option is specified and the expansion function using the following device file information is described.

- #pragma ioreg
- #pragma interrupt
- __set_il() function

3.1.2 Addition of -err_limit option

The “-err_limit” option that specifies the maximum number of output error messages (messages numbered in the 2000s) has been added. The maximum number of output error messages can be specified using this option.

- -err_limit=*num* (*num*: Decimal value from 15 to 50)

When this option is omitted, the output count is set to 15.
In PM plus, this option can be specified on the “Message” tab in the Compiler Options dialog box.

3.1.3 Addition of -err_file and +err_file options

The “-err_file” and “+err_file” options, which are used to output an error message to a file, have been added. The differences between the two options are as follows.

- -err_file=*file*: Saves the output file as *file* (overwrite)
- +err_file=*file*: Saves the output file *file* (additional save)

3.1.4 Addition of warning message

A warning message is now output for the following case.

- Integer value exceeding -2147483648

  A warning message is now output for an integer value exceeding -2147483648, because -2147483648 is a positive number (**unsigned int** type) according to the ANSI C language specifications.

  Message: W2172: constant out of range

3.1.5 Addition of -Xenum_type option

The “-Xenum_type” option that is used to specify the enumeration type and enumerator type has been added. The specifiable types are as follows.

- **char** type
- **unsigned char** type
- **short** type
- **unsigned short** type

When specification of -Xenum_type is omitted, it is handled as the **signed int** type. This option is specified in the following format.

-Xenum_type=*type* (*type*: char, uchar, short, or ushort)

In PM plus, this option is specified on the “C Language and Kanji” tab in the Compiler Options dialog box. If the enumerator exceeds the value range of each type, the following warning message is output.

Message: W2172: constant out of range

3.1.6 Enhancement of optimization

The following optimization functions have been enhanced or added in Ver. 2.70.

- Enhancement of loop optimization function
- Addition of optimization using **setf** instruction
- Acceleration of **switch** sort processing

3.1.7 Migration from 78K Series compiler

The -cc78k option, which enables the use of expansion functions compatible with the 78K Series C compiler "CC78Kx" has been added. Some functions have already been implemented in Ver. 2.60 and the "-Wf,-Xdescription=78k" option is used for them. In Ver. 2.70, all the expansion functions are specified by "-cc78k".

1. Support of __interrupt_brk function specifier

   In CA850, it is handled as the "__interrupt function specifier".

2. Output of warning message in response to #pragma directive

   - Change of section name

     The #pragma section directive of the 78K Series is handled as the #pragma section directive of the CA850. An illegal description error is output in response to a description that is not allowed in the CA850.

   - Change of module name

     A warning message is output in response to the #pragma name directive and this directive is ignored.

   - Memory manipulation

     The #pragma inline directive of the 78K Series is handled as the #pragma inline directive of the CA850. An illegal description error is output in response to a description that is not allowed in the CA850.

3. Output of warning message in response to identifier

   A warning message is output in response to an identifier (described in C language) for CC78Kx and the identifier is ignored.

   Message: W2761: unrecognized specifier '*ident*', ignored

   The identifier (described in C language) for which warning messages are output are as follows.

Item	Identifier
callt function specification	callt, __callt
noauto/norec function specification	noauto, norec
callf function specification	callf, __callf
Pascal function specification	__pascal
Bank function specification	__banked1 to __banked15
saddr area use specification	sreg, __sreg, __sreg1
Temporary variable specification	__temp

4. Output of error message in response to function call

   Processing is performed on the embedded functions for CC78Kx. The following warning message is output in response to the #pragma directive.

   Message: W2162: unrecognized pragma directive '*directive*', ignored

   The error message shown below is output if there is a #pragma directive and the corresponding function call. If there is not a #pragma directive, the function call is handled as an ordinary function call.

   Message: E2752: cannot call *function* function

   The error/warning is output in response to the following #pragma directives and functions.

Directive for specifying data insertion function #pragma opc __OPC
Directive for specifying 3-byte address reference/generation function #pragma addraccess FP_SEG, FP_OFF, MK_FP
Directive for specifying register direct reference function #pragma realregister __absa, __ashra, __clr1cy, __coma, __deca, __geta, __getax, __getcy, __inca, __nega, __not1cy, __rola, __rolca, __rora, __rorca, __seta, __setax, __setcy, __set1cy, __shla, __shra
Direct calling of self-programming sub-routine incorporated in firmware #pragma hromcall __hromcall, __hromcalla, __setsp, __FlashEnv, __FlashSetEnv, __FlashGetInfo, __FlashAreaBlankCheck, __FlashAreaErase, __FlashAreaIVerify, __FlashAreaPreWrite, __FlashAreaWriteBack, __FlashBlockBlankCheck, __FlashBlockPreWrite, __FlashBlockErase, __FlashBlockIVerify, __FlashBlockWriteBack, __FlashByteRead, __FlashByteWrite, __FlashWordWrite

3.1.8 Addition of Far Jump specification

When using "Far Jump" calling that enables branch to functions in the whole 32-bit space, if {all_function} is described in the function list file (file specified by the "-Xfar_jump" option), all function can now be called by Far Jump. In this case, all functions are targeted to be called even if another symbol name (function name) is specified.

3.1.9 Support of blank and S-JIS 2-byte character

Blank and S-JIS 2-byte characters can now be used for specifying a file name, folder name, or directory name. When using a blank, enclose the option that includes the folder or directory name with “ ” (double quotation marks).

Example:

```
> ca850 -devpath=c:\nectools32\dev -cpu F3107  "..\src files\main func.c"
```

When using PM plus, however, a file name containing a blank cannot be specified as a build target.

3.1.10 Modification of command files

When activating a command of an execution file such as ca850 or ld850, the following characters can now be handled as special characters in the command file in which an option or file name is described and passed, instead of directly specifying the option or file name as an argument of the command line.

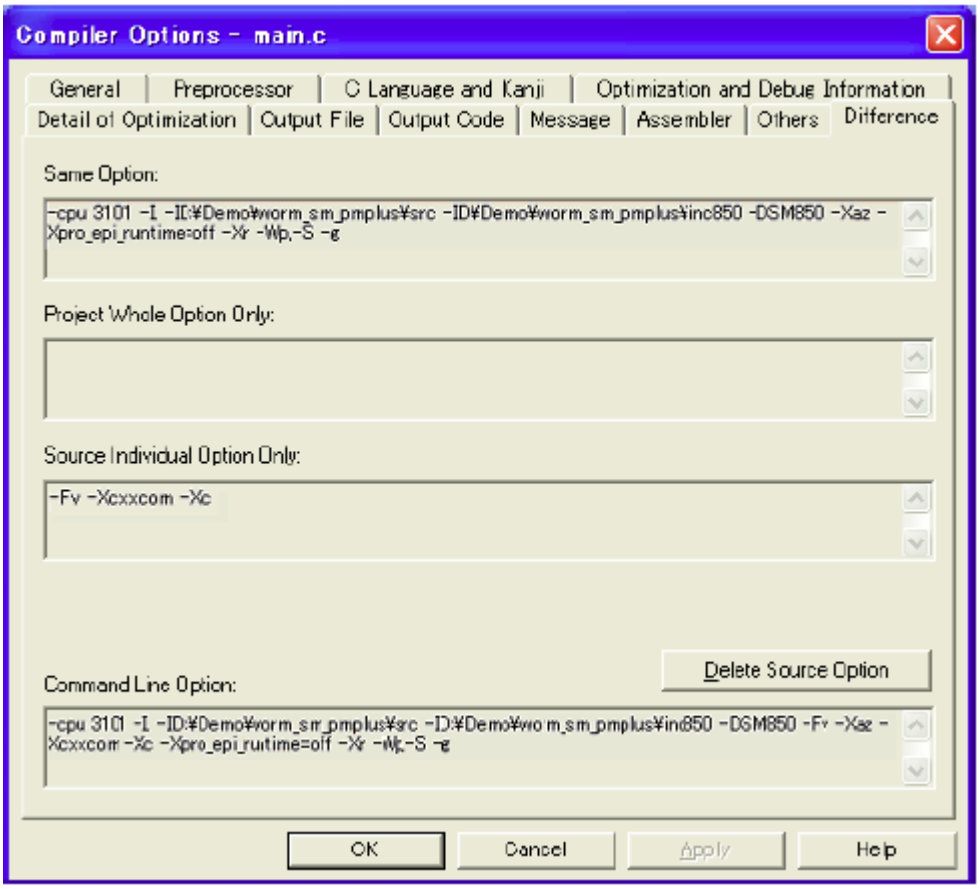
“ (double quotation mark)	The character string enclosed by “ ” (double quotation marks) is handled as a series of character strings.
# (sharp)	If specified at the top of a line, the line is handled as a comment.
^ (circumflex)	The character immediately after ^ (circumflex) is not handled as a special character. Use when not handling ” (double quotation mark) or # (sharp) as a special character.

In the as850, ar850, hx850, dump850, dis850, and romp850, only ” (double quotation mark) can be specified.

3.1.11 Displaying differences between source-specific options and project-common options

A function to display differences between assembler/compiler options specified for individual sources and assembler/compiler options specified for the whole project in PM plus has been added. In addition, the [Delete Source Option] button, which is used to delete the source-specific option has been added to this dialog box.

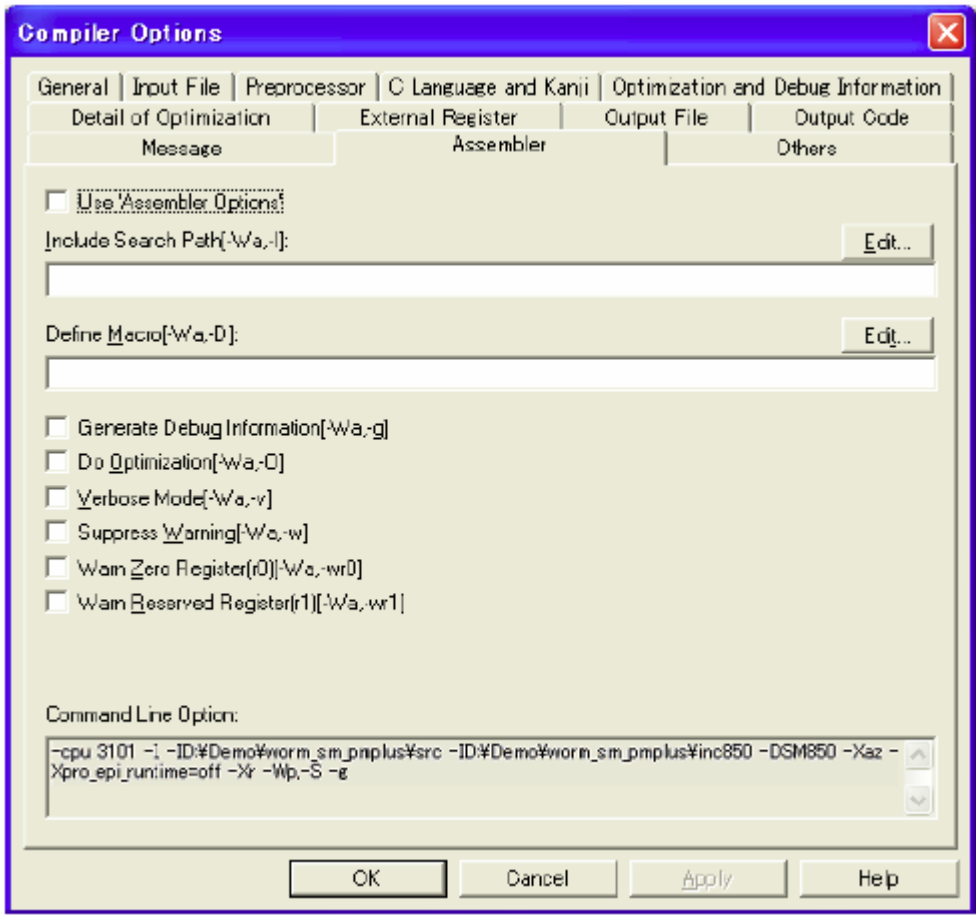
Figure 2-1. Difference Tab in Compiler Options Dialog Box



3.1.12 Assemble option specification after C-source compilation

An assembler option can now be specified for an assembler source created after C-source compilation in PM plus, on the Assembler tab in the Compiler Options dialog box. When using the options specified in the Assembler Options dialog box, check “Use ‘Assembler Options’” on this tab.

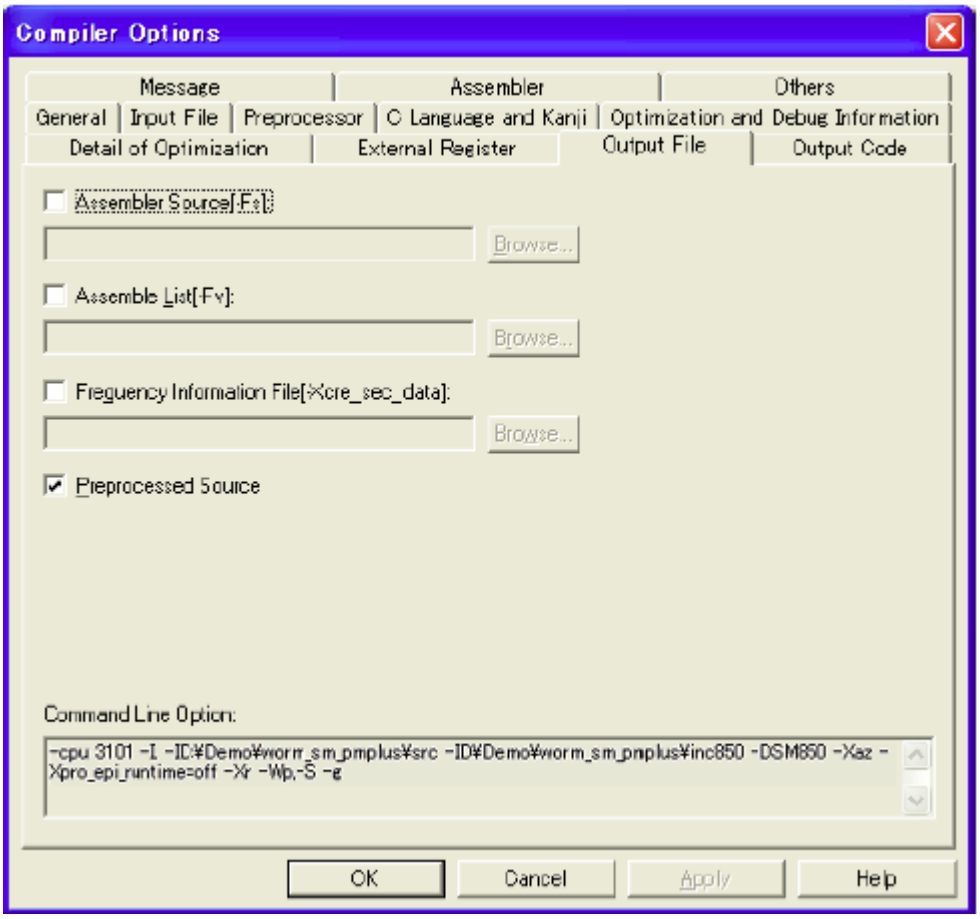
Figure 2-2. Assembler Tab in Compiler Options Dialog Box



3.1.13 Output of preprocessed source

A function to output a source on which only the preprocessing of the compiler is performed has been added to PM plus. Check “Preprocessed Source” on the “Output File” tab in the Compiler Options dialog box; a source file in which the extension “c” is substituted by “i” is then output.

Figure 2-3. Output File Tab in Compiler Options Dialog Box



3.1.14 Addition of assembler instruction expansion

The instruction expansion pattern when r0 is specified for dst has been added to the following assembler instructions.

Instruction Type	Instruction
Type 3	addi, mulhi
Type 7	satsubi
Type 12	andi, ori, xori

3.1.15 Far Jump of flash memory/external ROM re-link function

When re-linking the flash memory or external ROM by branching from the branch table in the flash memory to the actual function, branch was possible only within a 22-bit range (within ±1 MB). In Ver. 2.70, however, the Far Jump call (branch to the entire 32-bit space) can be used.

When the assembler quasi directive .ext_ent_size is described, Far jump is enabled by changing the entry size of the branch table used by the flash memory/external ROM re-link function. A Far Jump call is performed by specifying the entry size as 10 (8 in case of the V850E). See the Operation User's Manual for details.

3.1.16 Support of internal instruction RAM

The following functions have been added to the V850 Series products incorporating internal instruction RAM, such as the V850E/ME2.

1. Addition of _rcopy function types
   The "_rcopy" function, which is used to transfer from ROM to RAM during ROMization (when the -Xr option is specified), is now prepared for each number of bytes transferred. If the number of bytes transferred is fixed by the internal instruction RAM specification, use the corresponding _rcopy function (4 bytes in case of the V850E/ME2).

Transfer Size	Function Name
1 byte	_rcopy, _rcopy1
2 bytes	_rcopy2
4 bytes	_rcopy4

2. The alignment condition of each section is applied to the alignment condition of the "offset to the packed information of each section" in the rompsec section.

3. Addition of link directive file sample "v850def3.dir"
   The link directive file sample for V850 Series products with internal instruction RAM (V850E/ME2) has been added.

3.1.17 Addition of -rescan option

If the -rescan option is specified, in addition to the -l option that specifies the library to be referenced at linking, the library specified by the -l option is repeatedly searched until symbol resolution is complete. This option is used to prevent symbols from being unresolved due to searching based on the order of linking libraries. In PM plus, this option can be specified on the "Option" tab in the Linker Options dialog box.

3.1.18 Simplification of name specification of input link directive file

The file path can now be omitted when specifying the input file name of a link directive file. In Ver. 2.70, the input file can be specified as follows.

```
.text = $PROGBITS ?AX .text { file1.o };
```

In addition, specification of the object file in a library can now be omitted when specifying the library.

```
Lib = $PROGBITS ?AX .text { libfunc1.a };
```

3.1.19 Addition of link map file display

The following specifications have been added to the link map file to which mapping information is output as a result of linking.

1. Support of ROM/RAM capacity display

   "MEMORY ALLOCATION MAP" information has been added to the link map. The following information is displayed.

   - Output segment
   - Segment attribute
   - Address
   - Size (hexadecimal): Including alignment condition and alignment hole
   - Size (decimal): Including alignment condition and alignment hole

   By referencing this information, the user can confirm the size of allocated data and codes at once. An output example is shown below. This information is output at the top of a link map file.

```
                ********** MEMORY ALLOCATION MAP ***********

OUTPUT          SEGMENT         VIRTUAL         SIZE(16)        SIZE(10)
SEGMENT         ATTRIBUTE       ADDRESS

INT             RX              0x00000000      0x000003a4         932
SCONST          R               0x000002ab      0x0000178b        6027
TEXT            RX              0x00002000      0x0000fc80       64640
SIDATA          RW              0x03ffd800      0x000000f2         242
DATA            RW              0x03ffd980      0x00000b88        2952
```

2. Addition of -mo option

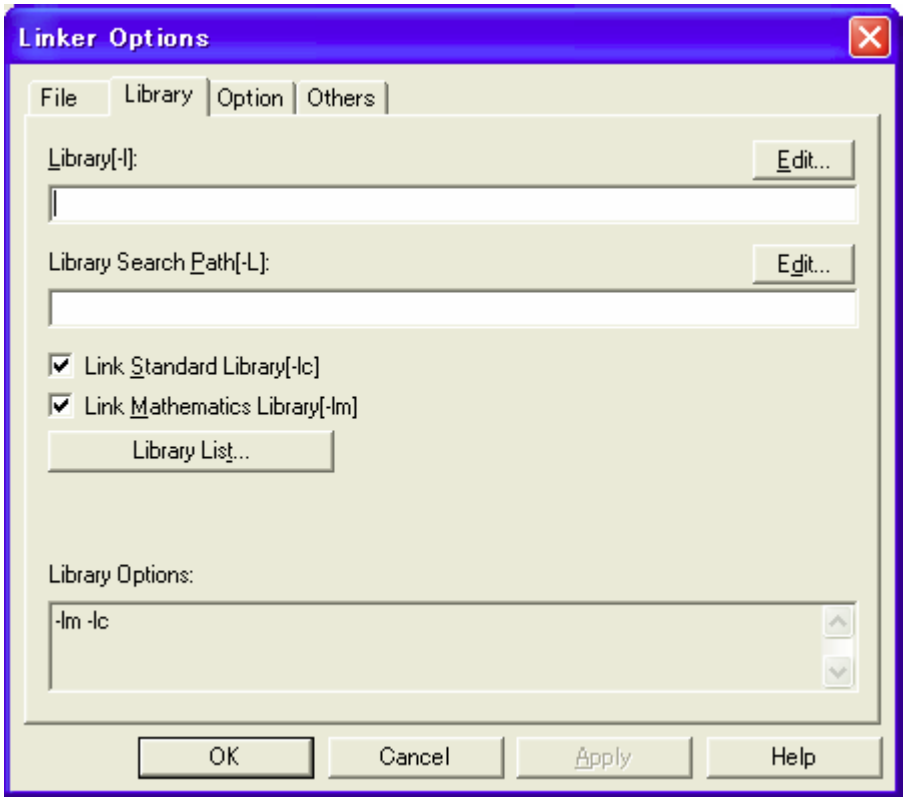
   This option is used to output the link map file in the Ver. 2.6x or earlier format.

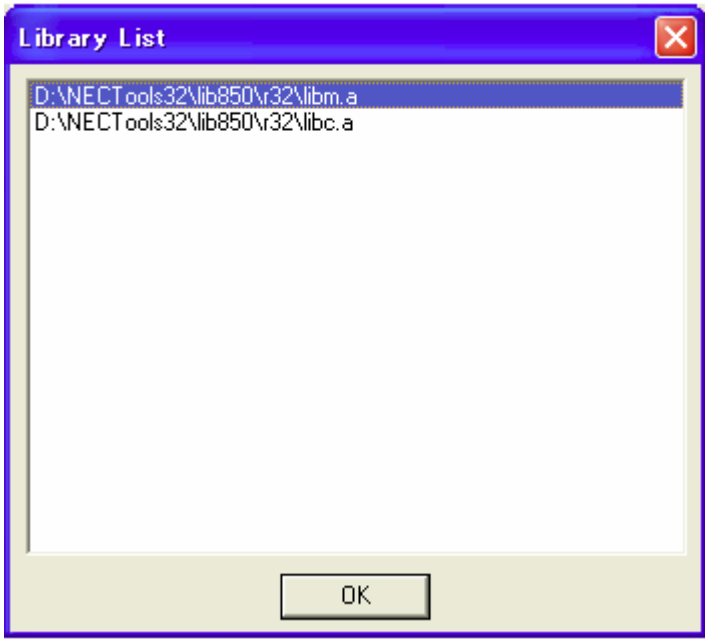
   -mo[=*mapfile*]

   When specification of *mapfile* is omitted, the link map file is output to "Standard Output". If both the -mo and -m options are specified, the -m option is ignored.

3.1.20 Reference library list display at linking

A list of libraries referenced at linking is now displayed in PM plus by clicking the [Library List…] button on the “Library” tab in the Linker Options dialog box.





3.1.21 Modification of ROMization processor “-m option”

Memory map information of an object created by the ROMization processor can now be output to a file; in V2.6x, only “Standard Output” (Output window output in case of PM plus) could be selected.

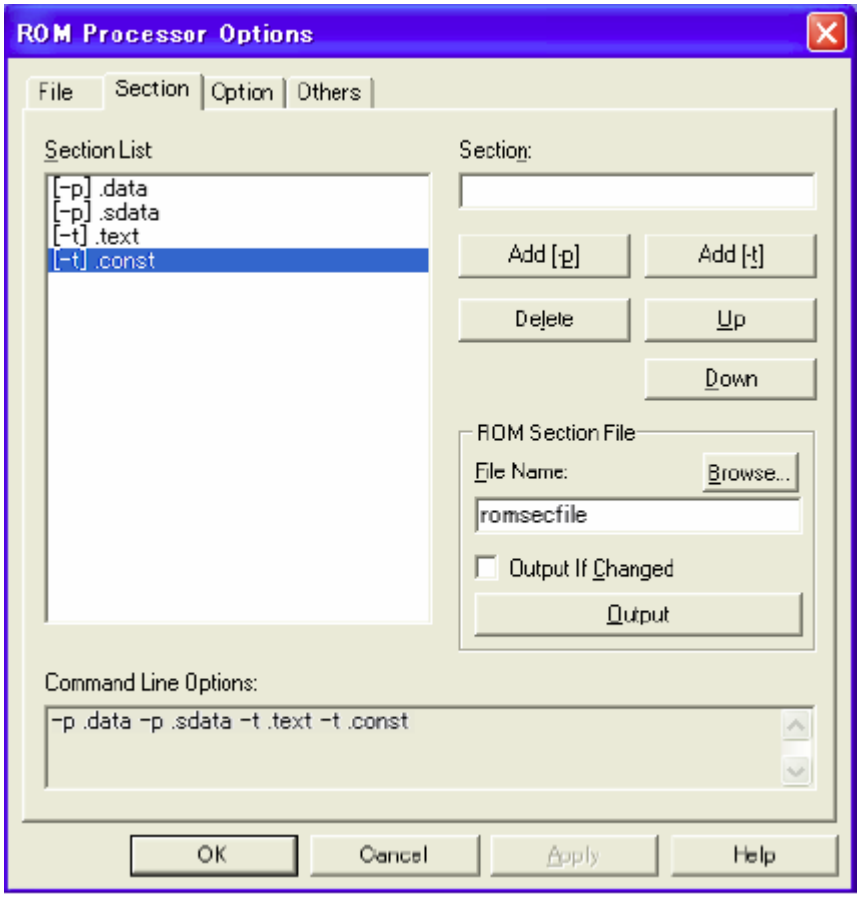
Specify output to a file by adding the output file name to the conventional -m option. An example of output to *mapfile* is as follows.

-m[=*mapfile*]

When specification of *mapfile* is omitted, the file is output to “Standard Output”. In PM plus, it is specified by inputting the output file name and checking “Memory Map File” on the “File” tab in the ROM Processor Options dialog box.

3.1.22 Changes in ROM Processor Options dialog box

Specification of the section name by setting the -p or -t option in the ROM Processor Options dialog box of PM plus is now easier than in the previous version. In addition, the ROM section file, which links (defines the macro by #define) the number and name of a packed section by specifying the _rcopy argument, can now be output. See the section of the ROMization processor in the Operation User’s Manual for detailed usage.



3.1.23 Modifications of hex converter

The following points have been modified in the hex converter (hx850).

1. Section sort

   In Ver. 2.70, sections are sorted in ascending address order. As a result, the -R option (option to specify sort in ascending address order) in Ver. 2.6x is no longer necessary. However, there is no problem in operation even if this option is specified in Ver. 2.70.

2. Modification of -U option

   The -U option that is used for specifying hex conversion of the ROM area was valid only for the Intel hex format, but in Ver. 2.70, it is also valid for the Motorola format.

   In addition, “2-byte specification” can now be selected for filling values in the vacant area.

   If both the -U option and -fT option are specified, the -U option is ignored.

3.1.24 Modification of section file format

The format of the frequency information file and section file has been changed as follows.

- The section name has been added to the variable information output to the frequency information file.
- Information of the variables that are only declared as extern is now not output to the frequency information file.
- The variable name output to the frequency information file or section file is enclosed by “ ” (double quotation marks).

```
“ file.c:function:var1”
“ file.c:var2”
“ var3”
```

The section file format of Ver. 2.6x or earlier is still valid.

3.1.25 Addition of section file generator function

The following functions have been added to the section file generator (sf850).

1. Addition of -size_tidata option

   This option is used to set the total size (bytes) of the tidata.word and tidata.byte sections assigned by the sf850.

   -size_tidata=*num* (*num*: Decimal number)

2. Addition of -size_tidata_byte option

   This option is used to set the size (bytes) of the tidata.byte section assigned by the sf850.

   -size_tidata_byte=*num*　(*num*: Decimal number)

3. Addition of -Xcs option

   When this option is specified, the variables assigned to the section for which the -O option (optimum allocation) is specified are excluded from the optimization target.

   -Xcs[=*section*]

   When *section* is not specified, it is handled as all the sections are specified.

4. Addition of -Xcv=*name* option

   When this option is specified, the variable specified by the -O option (optimum allocation) is excluded from the optimization target.

   -Xcv=*name*

3.1.26 Removal of restrictions up to Ver. 2.61

The following restrictions applicable to CA850 Ver. 2.61 or earlier have been removed in Ver. 2.70.

No. 1 Restriction on file name including blank character

[Description]

A file name or directory name including a blank character must not be used.

No.35 Restriction on **.option nomacro** quasi directive

[Description]

The mov instruction in the specified range of the **.option nomacro** quasi directive may result in a syntax error.

Example:

```
.text
.option nomacro
mov 0x12345678, r10
```

No.59 Restriction on internal ROM check by ROMization processor

[Description]

When using the ROMization processor (romp850), the following warning message is output when it should not be output, or not output when it should be output depending on the existence of a .text section or its allocated address. Note that the output object has no problem.

Message:

Warning: rompsec section overflowed highest address of target machine.

No.61 Restriction on multiple byte characters in a command file

[Description]

When a multiple byte character is used in the following command file, the CA850 fails to discriminate a descriptor in a character string, causing an error.

(1) as850 (Corrected in Ver. 2.60.)
(2) ar850 (Corrected in Ver. 2.50.)
(3) dis850 (Corrected in Ver. 2.70.)
(4) dump850 (Corrected in Ver. 2.70.)
(5) hx850 (Corrected in Ver. 2.50.)
(6) romp850 (Corrected in Ver. 2.60.)

No.88 Restriction on generating codes for complex expression

[Description]

An illegal code may be generated or a compile error may be output (E3228: illegal operand (must be register)) when executing a complex expression that includes many operations (at least four operators are included) and the expression satisfies the conditions shown below.

This bug may occur even if the number of operators is less than 4 when expressions are combined internally by optimization processing and satisfy the conditions shown below.
This bug more likely occurs when the number of temporary registers is insufficient upon code generation and the stack must be used instead. However, this bug does not always occur even if the conditions shown below are satisfied because it depends on the status of use of the registers and stack upon code generation.
The conditions under which this bug may occur are shown below. The conditions vary according to the CPU used (V850 core or V850E/V850ES core) and the version of the CA850.

Conditions:

(1) When using V850E/V850ES core and CA850 Ver.2.40/2.41, or when using V850 core and CA850 Ver. 2.40/2.41/2.50/2.60

(a) The operation includes a reference to a member that uses a “structure packing function”, and the member is

(a-1) a “2-byte type structure member allocated to an odd address”: See Example 1.

(a-2) a “4-byte type structure member allocated to an address that cannot be divided by 4”: See Example 2.

(b) The operation includes a reference from a structure pointer variable that uses a “structure packing function”, the packing value of the structure is 1, and the reference is to a 2-byte or 4-byte type member: See Example 1.

This bug may occur if any of (a-1), (a-2), or (b) is satisfied.

(2) When using V850E/V850ES core and CA850 Ver. 2.50/2.60

(a) The operation includes a reference to a member that uses a “structure packing function”, and:

(a-1) the member is a “2-byte or 4-byte type structure member allocated to an odd address”: See Example 1.

(a-2) the structure is allocated to the “tidata section”, and the member is a “2-byte type structure member allocated to an odd address” or “4-byte type structure member allocated to an address that cannot be divided by 4”: See Examples 1 and 2.

(b) The operation includes a reference from a structure pointer variable that uses a “structure packing function”, the packing value of the structure is 1, and the reference is to a 2-byte or 4-byte type member: See Example 3.

(c) The operation includes the include function __sasf() (the operation result of the __sasf function is used).

This bug may occur if any of (a-1), (a-2), (b), or (c) is satisfied.

Caution A “structure” includes the “bit field”. The “bit field members” are affected by “type”, not by “bit width”. For example, “int a:8;” is a 4-byte type member, and “short b:8;” is a 2-byte type member.

Example 1:

```
/* Example of member allocated to odd address */
#pragma pack(1)
struct {
    char pad1;
       ...
     /*
       Up to here, the total size of members from the start is an odd number
     */
    short mem; /* odd address */
};
```

Example 2:

```
/* Example of member allocated to address that cannot be divided by 4 (1) */
#pragma pack(1)
struct {
    char pad1;
       ...
     /*
       Up to here, the total size of members from the start is "multiple of 4 + 1"
     */
    int mem; /* odd address */
};

/* Example of member allocated to address that cannot be divided by 4 (2) */
#pragma pack(1)
struct {
    short pad1;
    char  pad2;
       ...
     /*
      Up to here, the total size of members from the start is "multiple of 4 + 3"
      */
    int mem;
};
```

Example 3:

```
/* Example when packing value is 1 and "2-byte or 4-byte type member is referenced
by structure pointer variable" */

#pragma pack(1)
struct {
    char c;
    int  i;
} *pst1;

int i1, i2, i3, i4;

int func() {
    return((~i1) << ((~i2)<<(((i3)+(i4)) << (pst1->i))));
                                              /* pst1->i is applicable */
}
```

No.89　Restriction on argument of inline function

[Description]

An illegal code is generated if a function on which inline expansion is performed (hereafter referred to as an inline function) satisfies the both following conditions.

- The address of an argument of the inline function is acquired, and the address is cast to a pointer whose type is different from the argument, and the contents of the argument are referenced.
- The argument above is not used by any other location.

The address resulting from cast becomes illegal where the inline function is expanded.

If there is a function for which #pragma inline is specified in the program, this bug may apply to the function.　In addition, a function is expanded inline and this bug may also occur depending on the optimization options “-Os” and “-Ot”, and the combination of options related to inline expansion.

Example of codes that cause bug:

```
#pragma inline subfunc

unsigned int s;

static void
subfunc(int i) {                /* this function is expanded inline */
    s = *((unsigned int*)&i);   /* cast */
}

void
func(int ii) {
    subfunc(ii);
}

void
main(void) {
    int t = 2;

    func(t);
    printf("%x\n", s);  /* references the contents of the argument */
}
```

3.1.27 Restrictions in Ver. 2.70

The restrictions applicable to CA850 Ver. 2.70 are as follows.

No.4 Restriction on precision during floating-point constant operation

[Description]

If a floating-point operation that may be inadvertently executed is described for compilation that involves casting to an integer, the precision drops very slightly, and the value may become illegal as a result of casting to an integer. No problem occurs if a floating point is handled as is.

Example:

```
(long)(1.12 * 100);
```

[Workaround]

Change the above statement as follows.

```
float  f = 1.12;
(long)(f * 100);
```

Or,

```
float  f;
(long)(f = 1.12 * 100);
```

[Correction]

Regard this as a restriction.

No.6 Restriction on structure type conditional operator for argument of function

[Description]

The correct branch code is not generated if a structure type conditional operator exists in an argument.

Example:

```
typedef struct {int i;}S;
S ss1, ss2;
int j;
void func(){
        func_call( ( j > 10 ) ? ss1 : ss2 );
}
```

[Workaround]

Change the above statement as follows.

```
if ( j > 10 ){
        func_call( ss1 );
}else {
        func_call( ss2 );
}
```

[Correction]

Regard this as a restriction.

No.7　Restriction on indirect calling of function

[Description]

If an expression that indirectly calls a function requires an offset, an internal compiler error occurs.

Message:

C2000:internal: gen_binary(): OP_CALL : left-child's operator is wrong

Example:

```
struct S {
        int dummy;
        int func_body[0x100];
} sobj;
void f(){
          ((void(*)())sobj.func_body)();
}
```

[Workaround]

Separate an offset calculation expression from a calling expression as follows.

```
void f(){
       void (*fp)() = (void(*)())&sobj.func_body;
       fp();
}
```

[Correction]

Regard this as a restriction.

No.8　Restriction on meaningless function definition

[Description]

An error is not output for a meaningless function definition.

Example:

```
typedef int INTFN();
INTFN f { return (0); }
```

[Workaround]

Avoid the coding that corresponds to this restriction.

[Correction]

Regard this as a restriction.

No.12　Restriction on extra ( ) in function declaration

[Description]

A syntax error is output for an extra ( ) in a function declaration.

Example:

```
typedef int Int;
void f1((Int));
```

[Workaround]

Modify the description as follows.

```
typedef int Int;
void f(Int);
```

[Correction]

Regard this as a restriction.

No.13 Restriction on type definition of structure that includes **extern**

[Description]

A syntax error is output for a type definition of a structure that includes **extern**.

Example:

```
extern struct tag { int i; };
```

[Workaround]

Remove **extern**.

```
struct tag { int i;};
```

[Correction]

Regard this as a restriction.

No.21 Restriction on section allocation

[Description]

In the **tidata** section allocation specification by the #pragma section directive or in "**char** type array of a structure" or "access to a **char** type member" specified by sf850 to be located in the tidata section, an error occurs in the linker if the displacement value of the sst instruction or sld instruction used to access the array or member is exceeded.

[Workaround]

Implement any of the following workarounds.

(1) Do not use the char member or **char** array of a structure that causes the error in the linker.

(2) Do not assign the char member or **char** array of a structure that causes the error in the linker to the **tidata** section.

[Correction]

Regard this as a restriction.

No.22 Restriction on section file with variable entity in assembly source

[Description]

In an application where the entity of a variable is in the assembly source and if that variable is referenced on the C source, an error occurs during linking if the section file is generated by sf850.

[Workaround]

Delete the variable in the assembly source from the section file.

[Correction]

Regard this as a restriction.

No.23 Restriction on section file with tentative definition of external variables of same name in multiple files

[Description]

If a section file is generated by sf850 when the tentative definition of external variables of the same name are in two or more files, symbols may be defined in duplicate during linking.

Message:

ld850: fatal error: simbol “_xxxx” multiply defined.

[Workaround]

If two or more tentative definitions of external variables of the same name exist, be sure to declare extern in the file that references the external variables.

[Correction]

Regard this as a restriction.

No.29 Restriction on specifying optimization option

[Description]

If the optimization level of the optimization option specified during compilation is increased, the phases that are executed during compilation (such as the optimization function and compilation function) increase. If the -Ot option is specified, the intermediate file created between these phases increases in size, causing a fatal error in some cases.

[Workaround]

Decrease the optimization level (by using -Os) and execute compilation.

[Correction]

Regard this as a restriction.

No.30 Restriction on object size at optimization

[Description]

When the optimization option is specified, the size of an object file including debug information may significantly increase.

[Workaround]

Either lower the optimization level or use the -g option, which outputs the debug information only to the file to be debugged.

[Correction]

Regard this as a restriction.

No.31 Restrictions on debugging at optimization

[Description]

When the optimization option is specified for debugging the source, the following restrictions apply.

a. When the value of a variable is referenced, a temporary value in the middle of calculation, not the correct value, may be obtained.
b. If part of an array, element of a structure, or user-defined pointer variable is assigned to a register, variables may be illegally displayed or modified in the variable window of the debugger.
c. If part of an array of an automatic variable or an element of structure is not used, the area may be deleted. In this case, variables may be illegally displayed or modified in the variable window of the debugger. The stack may be destroyed when a variable is modified.

[Workaround]

There is no workaround.

[Correction]

Regard this as a restriction.

No.33 Restriction on address of structure member

[Description]

If any of the conditions below apply when structure packing is performed, data access follows the data alignment of the device and the accessed address is masked. As a result, data will be missing or discarded by accessing the address of the structure member.

Conditions:

(1) The device used does not support misalign access

(2) The device used supports misalign access but misalign access is disabled

Example:

```
struct test {
        char c;  /* offset 0 */
        int  i;  /* offset 1-4 */
} test;
int *ip, i;
void func(){
        i = *ip;    /* Accessed from a masked address */
}
void func2(){
        ip = &(test.i);
}
```

[Workaround]

There is no workaround.

[Correction]

Regard this as a restriction.

No.34 Restriction on bit field

[Description]

If the width of a bit field is less than the type of a member when the bit field is accessed during structure packing, the bit field is read by the type of the member. Consequently, an area outside the object (area where there is no data) is also accessed. This access is usually executed correctly but it may be illegal if I/O is mapped.

Example:

```
struct S {
           int  x:21;
} sobj;  /* 3 bytes */
sobj.x = 1;
```

[Workaround]

There is no workaround.

[Correction]

Regard this as a restriction.

No.40 Restriction on referencing specific symbol and reserved symbol in C source

[Description]

Target-specific symbols such as __gp_DATA and reserved symbols such as _stext cannot be referenced in the

source.

[Workaround]

Do not use a target-specific symbol and reserved symbol in the C source.

[Correction]

Regard this as a restriction.

No.47 Restriction on output file path specification option of cross-reference tool

[Description]

Even if the folder to which the analysis result of a specified output file is to be output is specified by the path specification option (-o), the analysis result is not output to the specified folder but to the current folder (executed folder). If the -all option is specified, the -o option outputs the analysis result to the specified folder.

[Workaround]

Execute analysis in the folder to which the result is to be output.

[Correction]

Regard this as a restriction.

No.48 Restriction on output file path specification option of memory layout visualization tool

[Description]

Even if the folder to which the analysis result of a specified output file is to be output is specified by the path specification option (-o), the analysis result is not output to the specified folder but to the current folder (executed folder). If the -all option is specified, the -o option outputs the analysis result to the specified folder.

[Workaround]

Execute analysis in the folder to which the result is to be output.

[Correction]

Regard this as a restriction.

3.2 Changes in PM plus

This section explains changed points in PM plus.

3.2.1 Acceleration of processing

The speed of processing to close a workspace and build processing has been accelerated.

3.2.2 Addition of supported characters

A single-byte space and multi-byte characters can now be used for specifying the path (directory name), and multi-byte characters can be used for specifying the file name (single-byte spaces cannot be used for the file name).

3.2.3 Linking with .prw extension

The workspace file (extension: .prw) is now linked with PM plus. The relevant workspace can now be opened by PM plus by double-clicking the .prw file in Explorer.

3.2.4 Error message display in color

Error and warning messages that are output during build are now displayed in different colors.

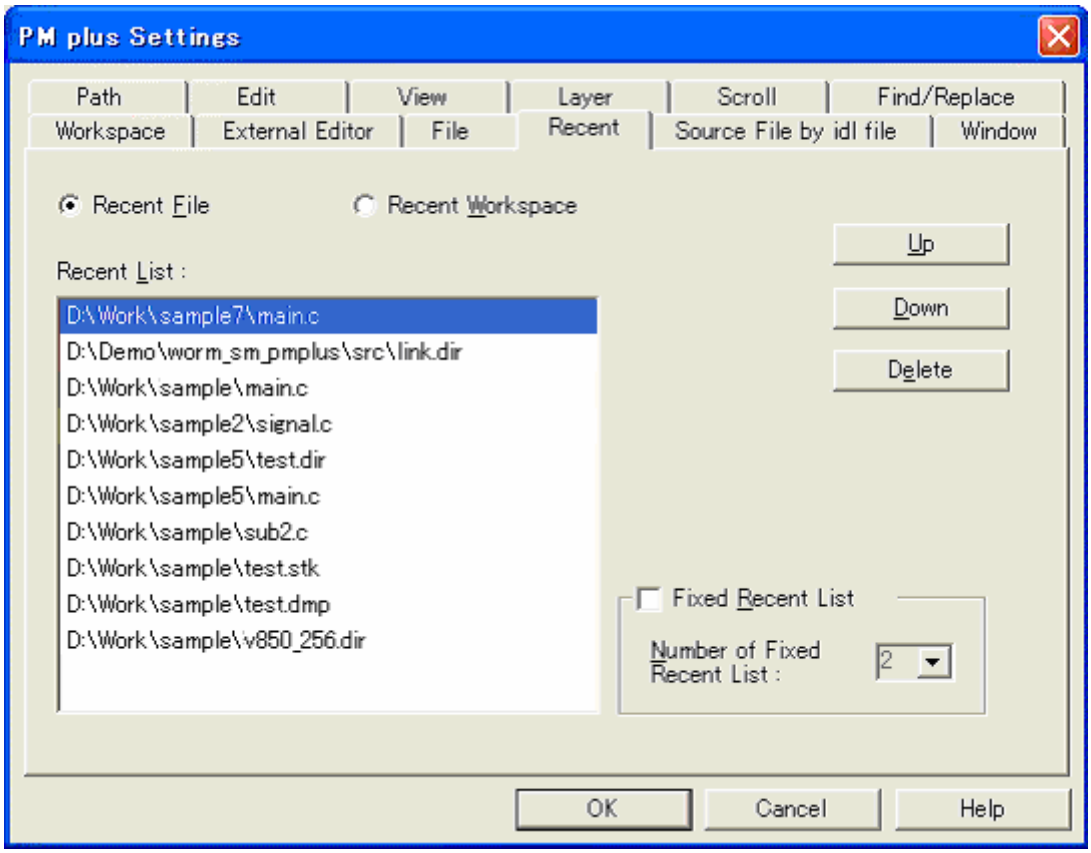
Error message	Displayed in Red
Warning	Displayed in Blue

3.2.5 Highlighted display of expansion SFR name

The expanded SFR name of the V850 Series is now highlighted in bold.

3.2.6 Editing of file/workspace history and specification of displayed list items

“Recent File” and “Recent Workspace” in the [File] menu can now be edited. The number of displayed items can be fixed. Consequently, the frequently opened files can now be specified quickly.



3.2.7 Enhancement of batch build function

The “batch build” function, which is used to build multiple projects sequentially, has been enhanced and the “batch re-build” function, which is used as a re-build function of “batch build”, has been added.

3.2.8 Link with Review-C

CA850 Ver. 2.70 is now linked with the NEC Communication Systems C-language program quality verification tool “Review-C”. Review-C can be activated from PM plus Ver. 5.20.
However, Review-C is not included with CA850 Ver. 2.70; it must be purchased separately.

3.2.9 Removal of restrictions

The following restrictions applicable to CA850 Ver. 2.61 (PM plus Ver. 5.10) has been removed in CA850 Ver. 2.70 (PM plus Ver. 5.20).

No.1 Restriction on replacement using regular expression in PM plus

[Description]

With PM plus, if replacement processing of a character string is performed using the regular expression “^”, which indicates the line head, and “$”, which indicates the end of the line, the linefeed codes that exist before and after the relevant character string are included in the replacement target.

Example:

When “Regular Expression” is selected in the Replace String dialog box and replacement of “^xyz” with “ABC” is executed, the line feed is also replaced as shown below.

```
------------------
abc
xyz
abc
------------------
↓
------------------
abcABC
abc
------------------
```

No.2 Restriction on square brackets in PM plus

[Description]

With PM plus, if a square bracket (“[” or “]”) is used in the file name, path name, or project group name, the workspace cannot be loaded and individual options cannot be set.