User's Manual

CB38 V.1.00

User's Manual

Custom Builder for Emulator Debugger PD38

Renesas Electronics
www.renesas.com

Rev.1.00 2003.05

For inquiries about the contents of this document or product, fill in the text file the installer generates in the following directory and email to your local distributor.

¥SUPPORT¥Product-name¥SUPPORT.TXT

Renesas Tools Homepage http://www.renesas.com/en/tools

1. Overview

1.1 Setting Up CB38

CB38 can be set up in the same way as for PD38. The procedure for setting up PD38 is detailed in the "Setup/Functional Outline" part of the PD38 V.2.00 User's Manual.

1.2 Features of CB38

CB38 provides an environment for using PD38's customize function to create exclusive script commands (hereafter called a "custom command program") or exclusive windows (hereafter called a "custom window program"). The custom command and custom window programs thus created by CB38 can be entered in PD38 to expand its functions.

The following shows the features of CB38:

1. The same user interface as available with PD38 is supported.
2. A development environment where programming, building, and debugging all are integrated is provided.
3. Creation of custom command and custom window programs is supported.
4. PD38's Register, Memory, Dump, and Script Windows are supported.

Each feature is detailed in the sections below.

1.3 Same user interface as available with PD38

CB38 uses the same graphical interface design as PD38, making it possible to use CB38 easily in the same way as for PD38.

1.4 Development environment where programming, building, and debugging all are integrated

CB38 allows you to control a series of operations from creating source files to building and debugging them. The windows supported by CB38 include Project, Message, Editor, Local, and Global Windows. Each of these windows allows you to manage projects, display the build result or other status, edit a source file, and display local and global symbols.

1.5 Creation of custom command and custom window programs

CB38 allows the type of program you are going to create to be specified from the dialog box that is opened when creating a project. In this way you can select the custom command or custom window program to be created.

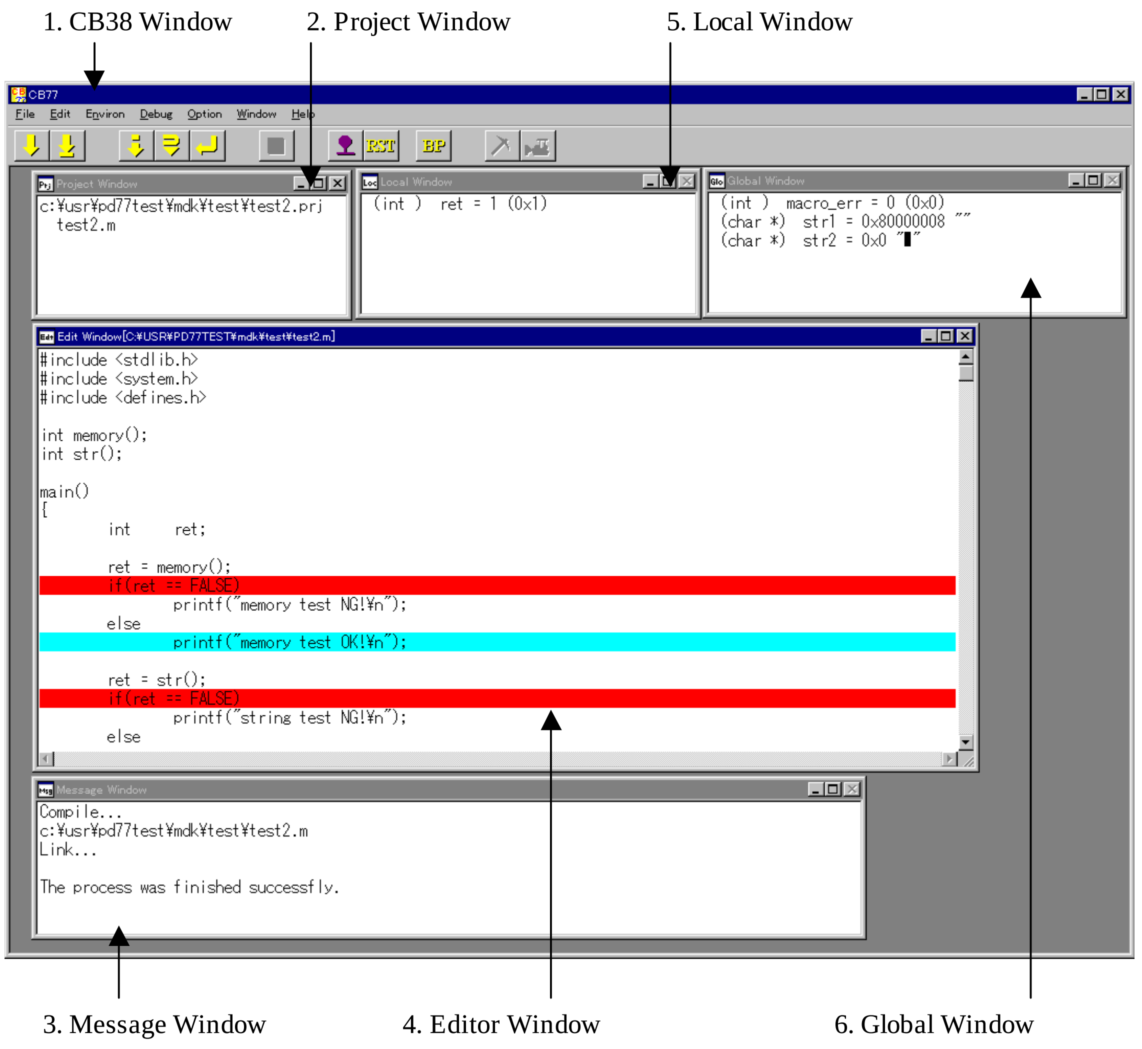
1.6 PD38's Register, Memory, Dump, and Script Windows

Among the windows available with PD38, CB38 supports the Register, Memory, Dump, and Script Windows. These windows can be used when creating custom command and custom window programs.

Note: The macro script commands cannot be used in the Script Window.

2. Function of Each Window

Figure 1 shows the window structure of CB38.



```
c:¥usr¥pd77test¥mdk¥test¥test2.prj
  test2.m
```

```
(int )  ret = 1 (0x1)
```

```
(int )  macro_err = 0 (0x0)
(char *)  str1 = 0x80000008 ""
(char *)  str2 = 0x0 "▮"
```

Edit Window[C:¥USR¥PD77TEST¥mdk¥test¥test2.m]

```
#include <stdlib.h>
#include <system.h>
#include <defines.h>

int memory();
int str();

main()
[
        int     ret;

        ret = memory();
        if(ret == FALSE)
                printf("memory test NG!¥n");
        else
                printf("memory test OK!¥n");

        ret = str();
        if(ret == FALSE)
                printf("string test NG!¥n");
        else
```

```
Compile...
c:¥usr¥pd77test¥mdk¥test¥test2.m
Link...

The process was finished successfly.
```

Figure 1. Window structure of CB38

The outline features and the functions of each window of CB38 are explained below.

2.1 CB38 Window

The CB38 Window is the main window of CB38. This is what opens first when you start up CB38.

2.1.1 Menu Bar

Tables 1 and 2 below show the menu bar structure of the CB38 Window.

Table 1. Structure of Menu Bar (CB38 Window) (1/2)

Menu item	Items on pull-down menu	Function
[F]ile	[N]ew	
	[S]ource/Header...	Create new source/header file.
	[P]roject...	Create new project.
	[O]pen...	Open source/project.
	[S]ave	Save source file.
	Save [A]s...	Save file after assigning a name.
	[C]lose	Close source file.
	E[x]it	Terminate CB38.
[E]dit	C[u]t	Delete specified range.
	[C]opy	Copy specified range to clipboard.
	[P]aste	Paste text from clipboard into position.
	[F]ind	Search for specified character string.
[E]nviron	[I]nit...	Open Init dialog box.
	[P]ath...	Open Path dialog box.
[D]ebug	[G]o	Execute Go command.
	[C]ome	Execute Come command.
	[S]tep	Execute Step command.
	[O]ver	Execute Over command.
	Retur[n]	Execute Return command.
	[A]nimate	Execute Animate command.
	[B]reak Point...	Open Break dialog box.
	Break Point	
	[S]et	Set or clear breakpoint.
	[L]ist...	Open Break dialog box.
	[R]eset	Reset program.
	[S]top	Stop program execution.
	B[u]ild	Built current project.
	R[e]Build	Rebuild current project.
[O]ption	Changed by window that has focus. (Refer to 3.2 and sections that follow.)	

Table 2. Structure of Menu Bar (CB38 Window) (2/2)

Menu item	Items on pull-down menu	Function
[W]indow	[C]ascade [T]ile [A]rrange Icon [R]egister Window M[e]mory Window [D]ump Window Scr[i]pt Window	Display windows one on top of another. Display windows side by side. Line up icons. Open PD38's Register Window. Open PD38's Memory Window. Open PD38's Dump Window. Open PD38's Script Window.
[H]elp	[I]ndex [A]bout...	Open table of contents of online help. Display version of CB38.

2.1.2 Tool Bar

Table 3 shows the tool bar structure of the CB38 Window.

Table 3: Structure of Tool Bar (CB38 Window)

Button	Function	Corresponding menu
↓	Execute Go command	[Debug]->[Go]
⤓	Execute Come command	[Debug] ->[Come]
↓	Execute Step command	[Debug] ->[Step]
⇓	Execute Over command	[Debug] ->[Over]
↵	Execute Return command	[Debug] ->[Return]
■	Stop program execution	[Debug] -> [Stop]
●	Set/clear breakpoint	[Debug] -> [Break Point] -> [Set]
RST	Reset program	[Debug] -> [Reset]
BP	Open Break dialog box	[Debug] -> [Break Point...]
⛏	Build project	[Debug] -> [Build]
⚒	Rebuild project	[Debug] -> [ReBuild]

2.2 Project Window

This window is used to manage the source files of the custom command and custom window programs created by CB38. The source file displayed in this window can be opened in the Editor Window by, for example, double-clicking the mouse button.

2.2.1 Menu Bar

Table 4 shows the menu bar structure of the Option menu of the Project Window.

Table 4. Menu Bar Structure of Option Menu (Project Window)

Menu item	Items on pull-down menu	Function
[O]ption	[S]et up... [A]dd File... [D]el File	Open Setup dialog box. Add source file to project. Delete source file from project.

2.3 Message Window

This window is used to display a compile or link error when building a project or other messages during debugging. These messages are initialized when you start building a project. When a compile error is displayed, point to the line in error and double- or single-click the mouse button to select it. Then choose [Option] -> [Jump] from the menu bar to display the corresponding source file in the Editor Window, with the cursor moved to the line in error.

2.3.1 Menu Bar

Table 5 shows the menu bar structure of the Option menu of the Message Window.

Table 5. Menu Bar Structure of Option Menu (Message Window)

Menu item	Items on pull-down menu	Function
[O]ption	[J]ump	Display lines in error.

2.4 Editor Window

This window is used to edit the source file. Multiple instances of this window can be opened at a time, with the source file name displayed on the title bar of each window. The Editor Window provides versatile editing functions, allowing you to input or delete characters, cut and paste to and from the clipboard, and load or save a file. During debugging, furthermore, a breakpoint line is shown in red and the next execution line is shown in blue. If a breakpoint line and the next execution line overlap, they are displayed in yellow.

2.4.1 Menu Bar

The Option menu of the Editor Window does not have any submenu.

2.5 Local Window

This window is used to display the local variables and their values of a function that corresponds to the program counter during debugging. This window is opened when you start debugging a program and is closed when you finish debugging.

2.5.1 Menu Bar

The Option menu of the Local Window does not have any submenu.

2.6 Global Window

This window is used to display global variables and their values during debugging. This window is opened when you start debugging a program and is closed when you finish debugging.

2.6.1 Menu Bar

The Option menu of the Global Window does not have any submenu.

3. Method for Creating a Program

This section explains how to use CB38 to create a custom command and a custom window program by using a simple program as an example.

3.1 Creating a Custom Command Program

The following shows the procedure for creating a custom command program by using CB38.

1. Create a new project for a custom command program.
2. Write a new source file.
3. Add the source file to the project.
4. Build the project.
5. Debug and correct the source file as necessary.
6. Repeat steps 5 and 6 until the program operates properly.

The table below shows specifications of the custom command program to be created in this section.

Program name	m_reset
Parameter	None
Function	Display program counter value before reset. Reset the target MCU. Display program counter value after reset.

3.1.1 Creating New Project for Custom Command Program

Choose [File]->[New]->[Project...] from the CB38 Window menu. The dialog box shown below will appear.

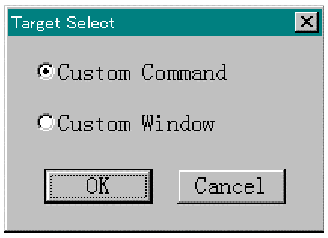


Figure 2. Target Select dialog box

Choose "Custom Command" and press the "OK" button.

A file selection dialog box will open, so input a project name and press the "Save" button. (A file name extension can be omitted.) The diagram below shows an example where "m_reset" is input for the name of the sample custom command program to be created in this section.

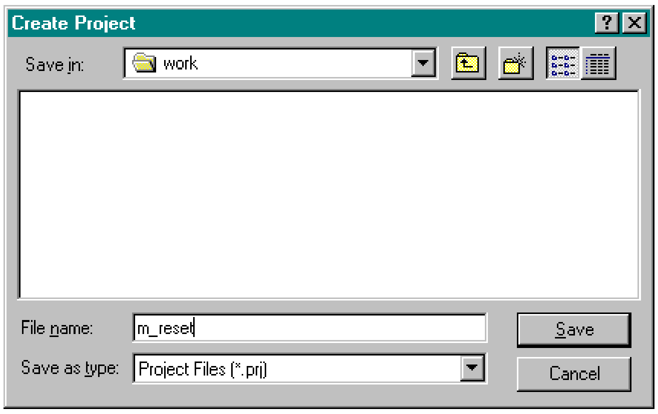


Figure 3. Dialog box for selecting a project name to be created

A Project Window showing the created project file name and a project setup dialog box are opened.

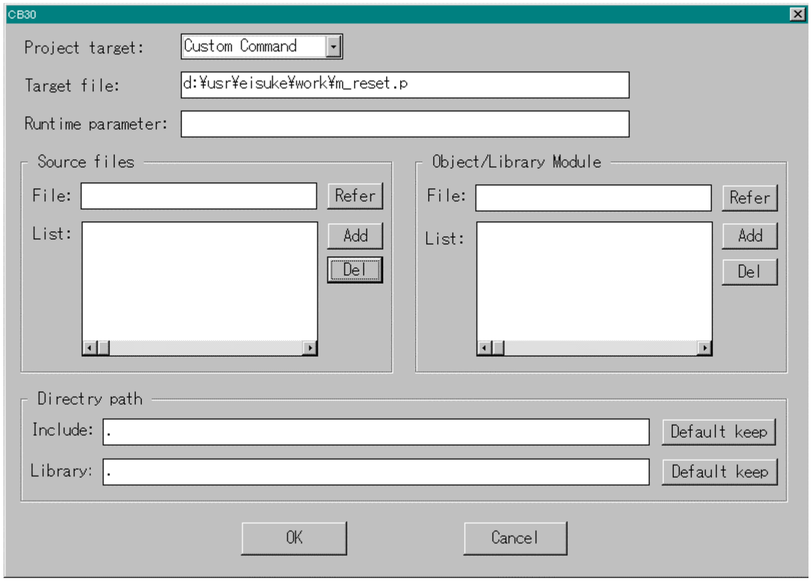


Figure 4. Setup dialog box

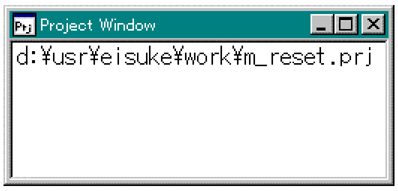


Figure 5. Project Window

The Setup dialog box can be opened from the Option menu of the Project Window to change its settings at any time you want. In this example, we only press the "Cancel" button on the Setup dialog box and leave it intact. For details on how to use the Setup dialog box, refer to Section 3.3, "Using Setup Dialog Box" on page 18.

Thus, with the above, a project file named "m_reset.prj" is created.

3.1.2 Creating New Source File

Choose [File]->[New]->[Source/Header...] from the CB38 Window menu. The Editor Window shown below will appear.

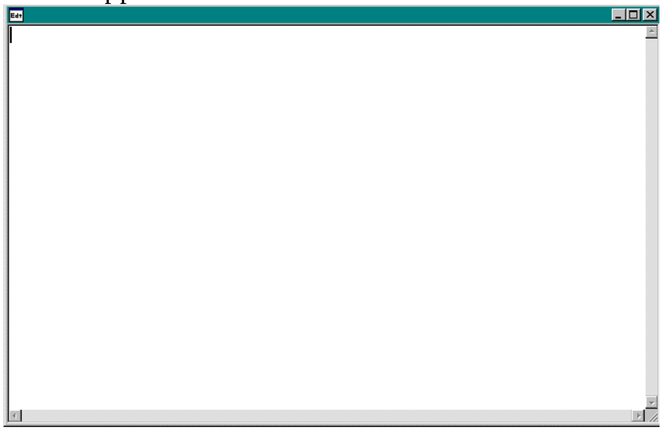


Figure 6. Blank Editor Window

Move focus to this Editor Window and choose [File]->[Save As...] from the CB38 Window menu to bring up a Save As dialog box. When this dialog box opens, input a file name and press the "Save" button. Specify ".m" for the source file name extension.

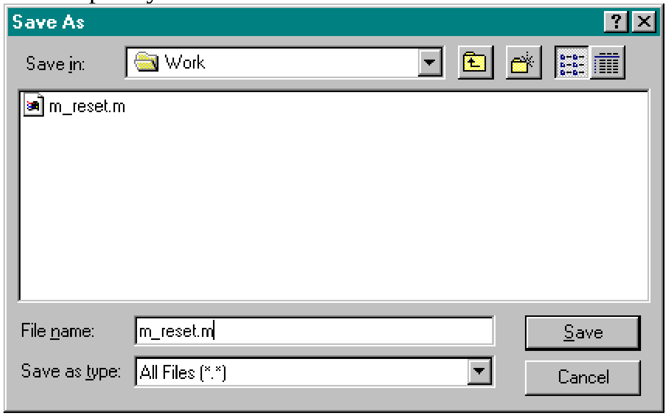


Figure 7. Save As dialog box

The name you have input in the Save As dialog box is displayed on the title bar of the Editor Window.

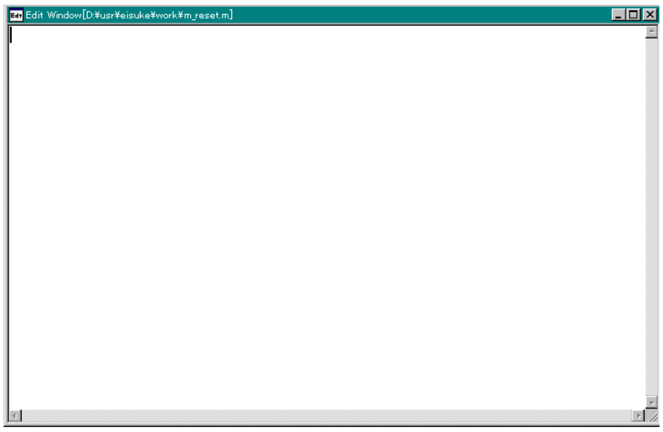
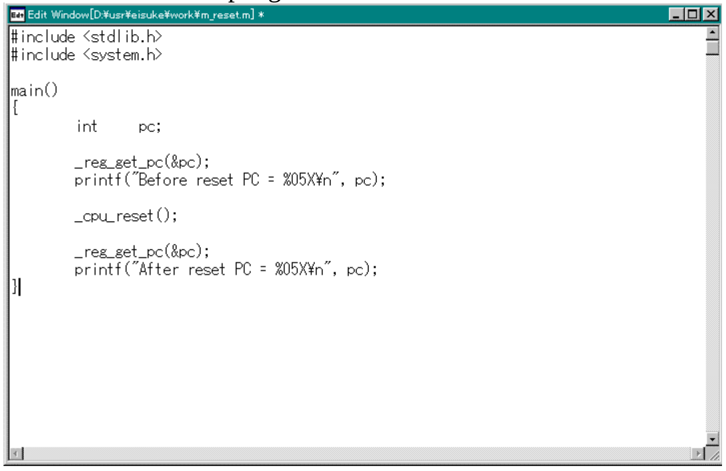


Figure 8. Editor Window with its name shown on title bar

Write a custom command source program in this Editor Window.



```
#include <stdlib.h>
#include <system.h>

main()
{
        int     pc;

        _reg_get_pc(&pc);
        printf("Before reset PC = %05X¥n", pc);

        _cpu_reset();

        _reg_get_pc(&pc);
        printf("After reset PC = %05X¥n", pc);
}
```

Figure 9. Editor Window with a source program written in it

For details about programming language specifications, refer to Section 4, "Programming Language Specifications" on page 23.

For details about library function specifications, refer to Section 5, "Reference" on page 24.

The asterisk (*) at the end of the file name on the title bar indicates that changes have been made to this file.

Thus, with the above, a custom command source file named "m_reset.m" is created.

3.1.3 Add Source File to Project

To build the source file created in the preceding section, we need to add it to a project. Choose [Option]->[Add File...] from the Project Window menu to bring up an "Add in source" dialog box. When this dialog box opens, choose the file name you want to be added to a project and press the "Open" button. The source file name thus added is displayed in the Project Window.

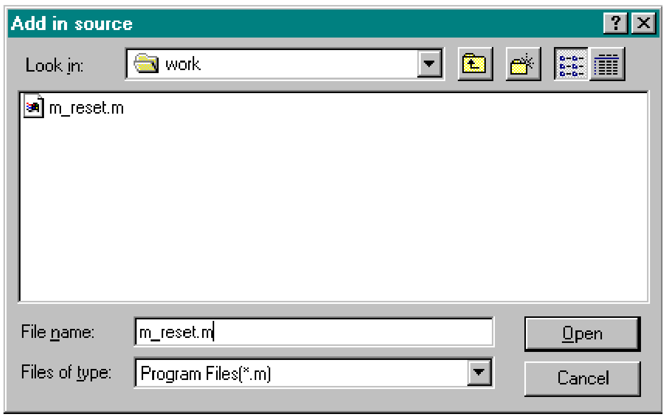
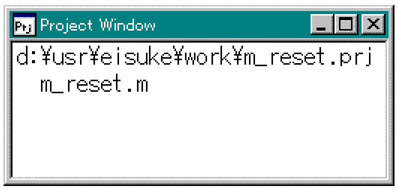


Figure 10. "Add Source" dialog box



```
d:¥usr¥eisuke¥work¥m_reset.prj
  m_reset.m
```

Figure 11. Project Window with a source file added

Thus, with the above, the source file "m_reset.m" is added to the project.

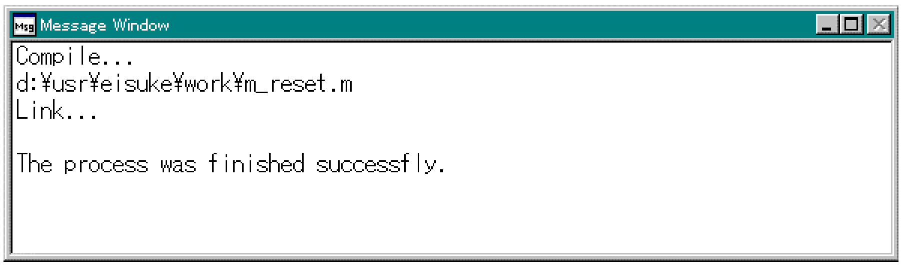
You also can add source files to a project using the Setup dialog box. For details on how to use the Setup dialog box, refer to Section 3.3, "Using Setup Dialog Box" on page 18.

3.1.4 Building a Project

The operation to create a custom command program and a custom window program file by processing the source files added to a project is referred to as "build" or "rebuild." The difference between "build" and "rebuild" is that among the source files added to a project, only those which have been modified since a program file was created previously are processed in the former, whereas all of the source files added to a project are processed in the latter.

To execute Build, choose [Debug]->[Build] from the CB Window menu or press the Build button on the tool bar.

To execute Rebuild, choose [Debug]->[ReBuild] from the CB Window menu or press the Rebuild button on the tool bar.

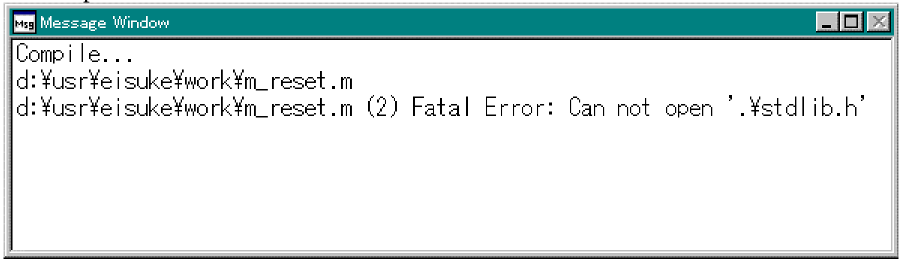


```
Compile...
d:¥usr¥eisuke¥work¥m_reset.m
Link...

The process was finished successfly.
```

Figure 12. Message Window when succeeded in building

Thus, with the above, a custom command program file is generated by CB38 providing that no error is found in the source program and in settings of the Setup dialog box.

In this example, the include file and library file search paths remain set to the default value (current directory) because we only pressed the "Cancel" button in the Setup dialog box that opened when creating a project. Therefore, if the project was built following the process described above, a message will be displayed in the Message Window indicating that include files cannot be opened.



```
Compile...
d:¥usr¥eisuke¥work¥m_reset.m
d:¥usr¥eisuke¥work¥m_reset.m (2) Fatal Error: Can not open '.¥stdlib.h'
```

Figure 13. Message Window when an error occurred when building

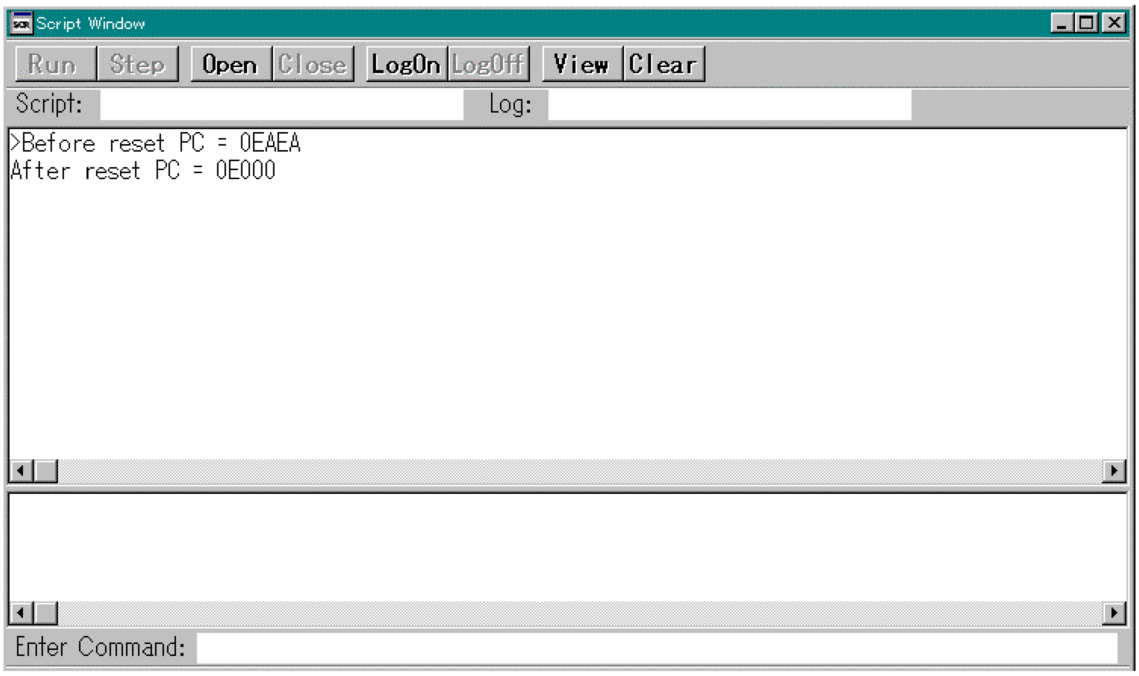
In this case, click on the error message line displayed in the Message Window and then choose [Option]->[Jump] or double-click on the error message line. The corresponding source line will be displayed in the Editor Window, with the cursor moved to that line.

In the example here, the Build operation can be successfully executed by setting the include file and library file search paths properly.

For details on how to use the Setup dialog box, refer to Section 3.3, "Using Setup Dialog Box" on page 18.

3.1.5 Execution Example of Custom Command Program

The following shows an execution example of the m_reset command program that was created in the example above. To execute a command program, press the Go button on the CB38 Window tool bar.



```
>Before reset PC = 0EAEA
After reset PC = 0E000
```

Figure 14. Execution example of custom command program "m_reset.p"

In this example, you will see that the PC address before a reset is EAEAH and the PC address after a reset is E000H.

Output from custom command programs are fed into the Script Window. Therefore, if the Script Window is not open, there is no means of verifying output from custom command programs.

3.2 Creating a Custom Window Program

The following shows the procedure for creating a custom window program by using CB38.

1. Create a new project for a custom window program.
2. Edit the framework source file generated by CB38.
3. Build the project.
4. Debug and correct the source file as necessary.
5. Repeat steps 3 and 4 until the program operates properly.

The table below shows specifications of the custom window program to be created in this section.

Program name	dump1000
Function	Dump 128 bytes beginning with address 1000H.

3.2.1 Creating New Project for Custom Window Program

Choose [File]->[New]->[Project...] from the CB38 Window menu. The dialog box shown below will appear.

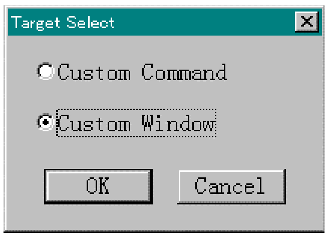
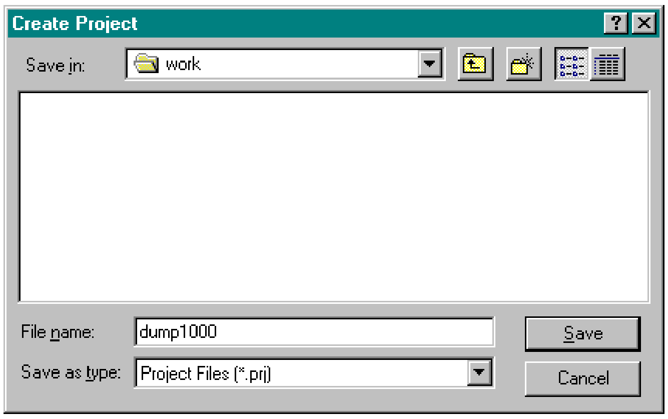


Figure 15. Target Select dialog box

Choose "Custom Window" and press the "OK" button.

A file selection dialog box will open, so input a project name and press the "Save" button. (A file name extension can be omitted.) The diagram below shows an example where "dump1000" is input for the name of the sample custom window program to be created in this section.


Create Project

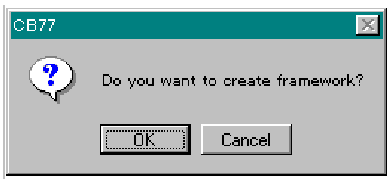
Save in: work

File name: dump1000

Save as type: Project Files (*.prj)

Figure 16. Dialog box for selecting a project name to be created

When the dialog box prompting for your confirmation of whether or not to create framework shown below appears, enter "Yes".

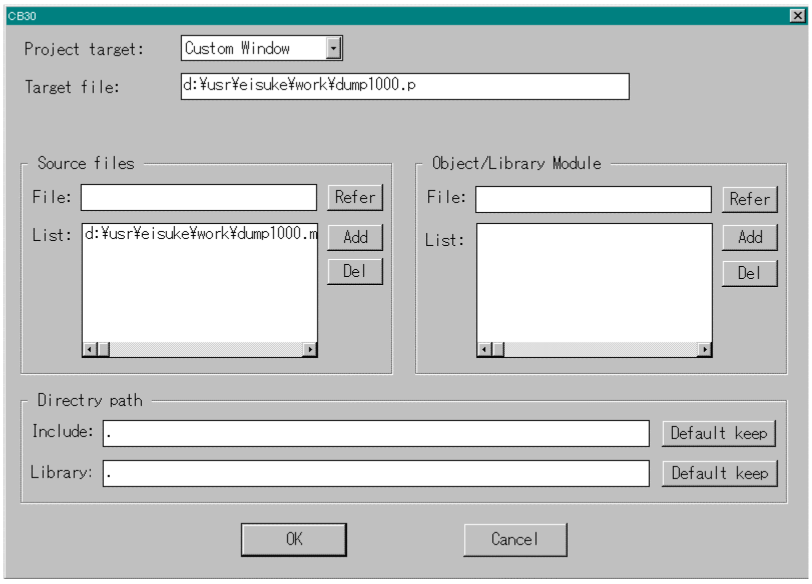

CB77

Do you want to create framework?

Figure 17. Dialog box for confirmation of framework generation

If you enter "No" here, CB38 does not automatically create framework.

A Project Window showing the created project file name and a project setup dialog box are opened.


CB30

Project target: Custom Window

Target file: d:¥usr¥eisuke¥work¥dump1000.p

Source files

File:

List: d:¥usr¥eisuke¥work¥dump1000.m

Object/Library Module

File:

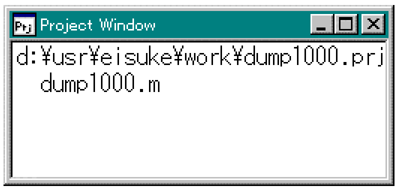
List:

Directry path

Include: .

Library: .

Figure 18. Setup dialog box


Project Window

```
d:¥usr¥eisuke¥work¥dump1000.prj
  dump1000.m
```

Figure 19. Project Window

The Setup dialog box can be opened from the Option menu of the Project Window to change its settings at any time you want. In this example, we only press the "Cancel" button on the Setup dialog box and leave it intact. For details on how to use the Setup dialog box, refer to Section 3.3, "Using Setup Dialog Box" on page 18.

When creating a project for a custom window program, a framework source file is automatically generated by CB38. In this example, the file "dump1000.m" is automatically generated. Programming of a custom window program is accomplished by editing this framework source file.

Thus, with the above, a project file "dump1000.prj" and a framework source file "dump1000.m" are created.

3.2.2 Editing Automatically Created Framework Source File

The framework source file automatically created by CB38 contains a description of the handle functions that correspond to window events.

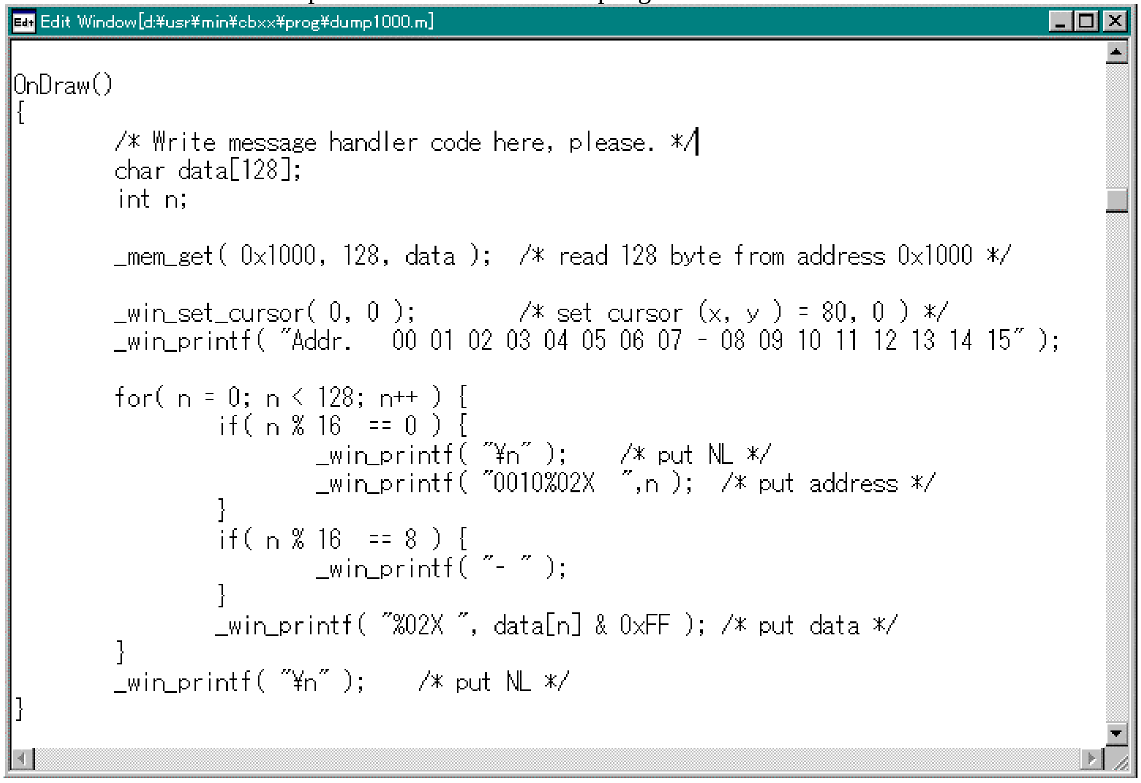
For details about handle functions, refer to Section 5.4, "Handle Functions for Custom Window" on page 85.

Two handle functions are treated in the example here: OnDraw and OnEvent. The OnDraw function is called when an area hidden in some other window need to be displayed. The OnEvent function is called when a change in debugger status is required such as when the target's memory value has been modified.

When the OnDraw function is called, dump1000 gets 128 bytes of memory values starting from address 1000H and convert them into character strings for display in window. To write this series of processing, edit the internal statements of the OnDraw function. Furthermore, when the OnEvent function is called, dump1000 calls the OnDraw function to update the window display.

Note: Do not delete the functions written in the framework source file. Loss of any function in this file makes it impossible to build a project correctly. There is no limit to the functions that can be added to the file.

The diagram below shows an Editor Window displaying the OnDraw function that has been edited for the "dump1000" custom window program.



Edit Window[d:¥usr¥min¥cbxx¥prog¥dump1000.m]

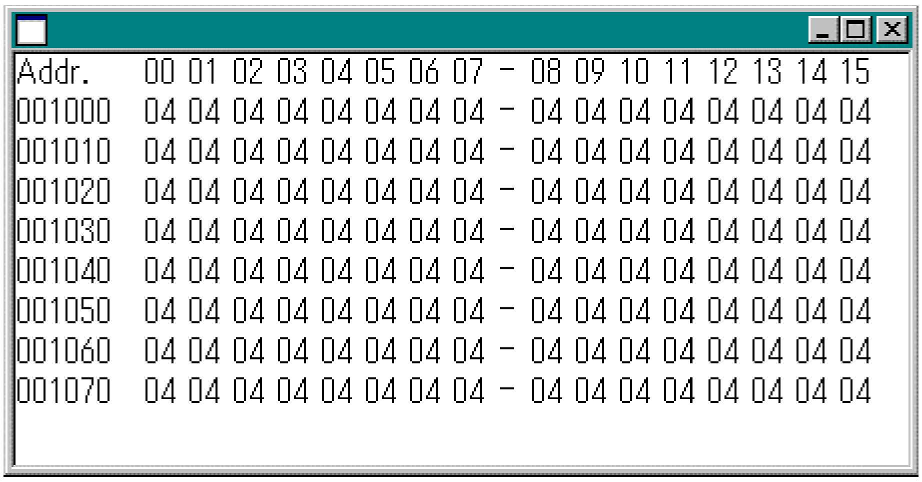
```
OnDraw()
{
        /* Write message handler code here, please. */
        char data[128];
        int n;

        _mem_get( 0x1000, 128, data );  /* read 128 byte from address 0x1000 */

        _win_set_cursor( 0, 0 );        /* set cursor (x, y ) = 80, 0 ) */
        _win_printf( "Addr.   00 01 02 03 04 05 06 07 - 08 09 10 11 12 13 14 15" );

        for( n = 0; n < 128; n++ ) {
                if( n % 16  == 0 ) {
                        _win_printf( "¥n" );    /* put NL */
                        _win_printf( "0010%02X  ",n );  /* put address */
                }
                if( n % 16  == 8 ) {
                        _win_printf( "- " );
                }
                _win_printf( "%02X ", data[n] & 0xFF ); /* put data */
        }
        _win_printf( "¥n" );    /* put NL */
}
```

Figure 20. Editor Window displaying OnDraw function for dump1000

The method for building a project for a custom window program is the same as used for custom command programs. Refer to Section 3.1.4, "Building a Project" on page 11.

3.2.3 Execution Example of Custom Window Program

The following shows an execution example of the dump1000 window program that was created in the example above. To execute a window program, press the Go button on the CB38 Window tool bar.



```
Addr.   00 01 02 03 04 05 06 07 - 08 09 10 11 12 13 14 15
001000  04 04 04 04 04 04 04 04 - 04 04 04 04 04 04 04 04
001010  04 04 04 04 04 04 04 04 - 04 04 04 04 04 04 04 04
001020  04 04 04 04 04 04 04 04 - 04 04 04 04 04 04 04 04
001030  04 04 04 04 04 04 04 04 - 04 04 04 04 04 04 04 04
001040  04 04 04 04 04 04 04 04 - 04 04 04 04 04 04 04 04
001050  04 04 04 04 04 04 04 04 - 04 04 04 04 04 04 04 04
001060  04 04 04 04 04 04 04 04 - 04 04 04 04 04 04 04 04
001070  04 04 04 04 04 04 04 04 - 04 04 04 04 04 04 04 04
```

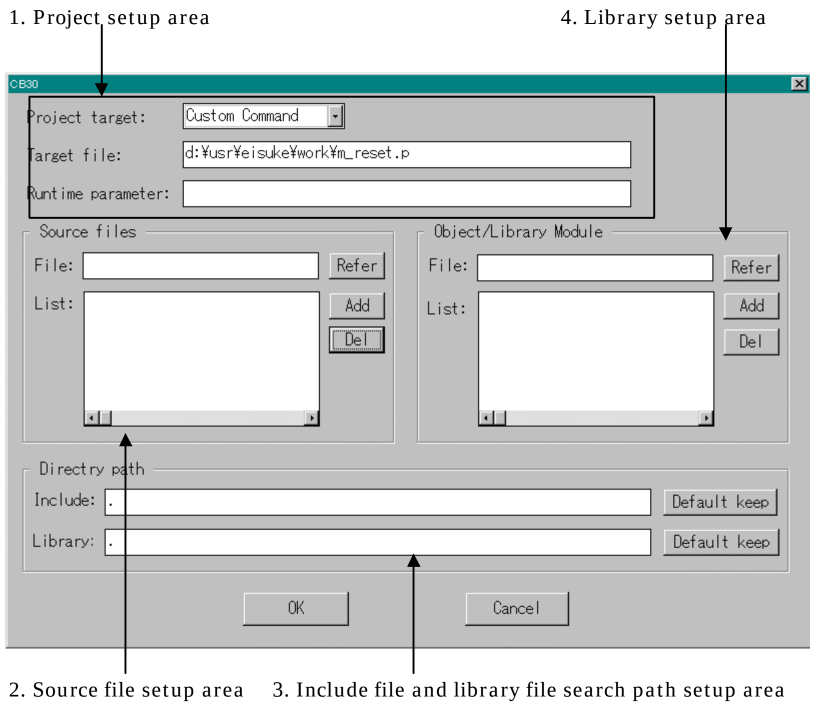
Figure 21. Execution example of custom window program "dump1000.p"

In this example, you will see that 128 bytes beginning with address 1000H are displayed in dump form.

When an area hidden in some other window need to be displayed, a custom window program calls the OnDraw function; when the debugger status need to be updated such as when the target memory contents have been changed, it calls the OnEvent function. Therefore, the dump1000 custom window program has its display automatically updated when a hidden part is displayed or target memory contents are changed.

3.3 Using Setup Dialog Box

The Setup dialog box is provided for setting up a project. This dialog box is opened by choosing [Option]->[Set up...] from the CB38 Window menu or double-clicking on the project file name displayed in the Project Window.

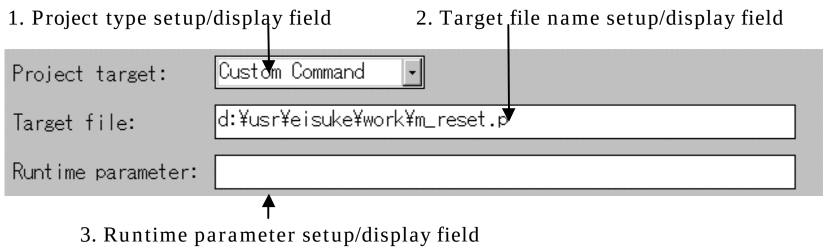


Project target: Custom Command
Target file: d:¥usr¥eisuke¥work¥m_reset.p
Runtime parameter:
Include: .
Library: .

Figure 22. Structure of Setup dialog box

3.3.1 Project Setup Area

This area is comprised of the following three fields:


1. Project type setup/display field
2. Target file name setup/display field
3. Runtime parameter setup/display field

Project target: Custom Command
Target file: d:¥usr¥eisuke¥work¥m_reset.p
Runtime parameter:

Figure 23. Structure of project setup field

3.3.1.1 Project Type Setup/Display Field

One of the following two project types can be set here.

Custom Command	Create custom command program.
Custom Window	Create custom window program

The project type you have set is displayed in this field

The startup routines and libraries that will be combined during building are selected depending on the project type you choose for the program to be created. A change of the project type only affects the selection of the startup routines and libraries that will be combined during building.

3.3.1.2 Target File Name Setup/Display Field

Set the program file name here that you want to be created when building.

The file name you have set is displayed in this field.

3.3.1.3 Runtime Parameter Setup/Display Field

This field appears when you specified "Custom Command" for the project type. Set the parameters in this field that you want to be passed when debugging a custom command program. The parameters set here are passed to the arguments “argc” and “argv” of the main() function in the following manner:

argc	Number of parameters
argv	Pointer array address that contains pointers to areas where character strings specified in parameters are stored

The parameters you have set are displayed in this field.

3.3.2 Source File Setup Area

This area is comprised of the following five fields:

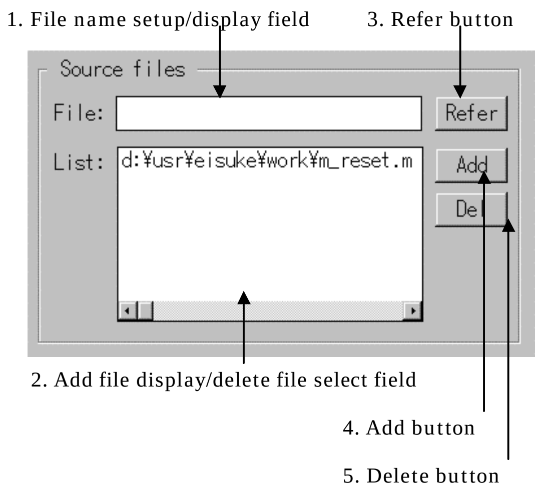

1. File name setup/display field
3. Refer button
Source files
File:
Refer
List:
d:¥usr¥eisuke¥work¥m_reset.m
Add
Del
2. Add file display/delete file select field
4. Add button
5. Delete button

Figure 24: Structure of source file setup area

3.3.2.1 File Name Setup/Display Field

Set a source file name in this field that you want to be added to a project.

The source file set here is added to a project as you press the "Add" button and the source file name is displayed in the add file display/delete file select field.

The source file names added to a project are listed as you press the "Add" button.

3.3.2.2 Add File Display/Delete File Select Field

The source file names added to a project are listed in this field.

An unnecessary source file can be deleted from a project by selecting its file name in this field by clicking on it with the mouse and pressing the "Delete" button.

3.3.2.3 Refer Button

The source file names added to a project are listed in this field.

An unnecessary source file can be deleted from a project by selecting its file name in this field by clicking on it with the mouse and pressing the "Delete" button.

3.3.2.4 Add Button

This button adds the source file that is entered in the file name setup/display field to a project.

When you add a source file, CB38 checks to see if the file exists. If the specified source file does not exist or has already been added to a project, no file is added.

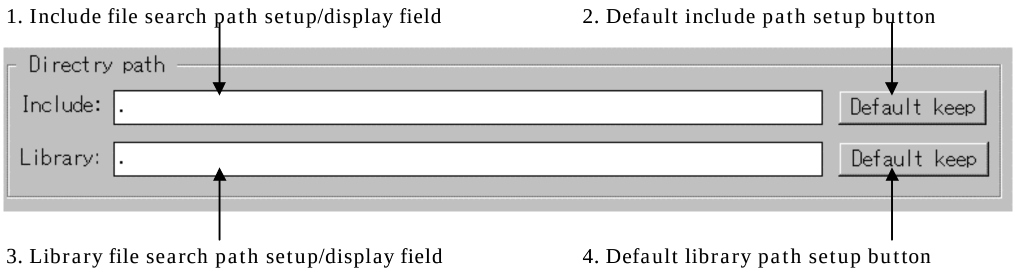
3.3.2.5 Delete Button

This button deletes the source file from a project that you have selected by clicking on it with the mouse in the add file display/delete file select field.

No file is deleted unless there is any source file selected.

3.3.3 Include File and Library File Search Path Setup Area

This area is comprised of the following four fields:


1. Include file search path setup/display field
2. Default include path setup button
Directry path
Include:
Default keep
Library:
Default keep
3. Library file search path setup/display field
4. Default library path setup button

Include: .

Library: .

Figure 25. Structure of include file and library file search path setup area

3.3.3.1 Include File Search Path Setup/Display Field

Set the directory in this field that you want to be searched for a file when inclusion of a file is specified by ***#include <filename>*** in the source file.

Normally, specify a directory where system include files are stored.

The system include files are installed in the “include” directory that is located below the directory where CB38 is installed.

The include file search path you have set is displayed in this field.

3.3.3.2 Default Include Path Setup Button

This button sets the directory that is set in the include file search path setup/display field as the default path to be used for CB38 when creating a new project.

When you create a new project with CB38 after setting the default path with this button, the directory you have set is used as the include file search path.

3.3.3.3 Library File Search Path Setup/Display Field

Set the directory in this field that you want to be searched for a library file to be linked when building a project.

Normally, specify a directory where system library files are stored.

The system library files are installed in the “lib” directory that is located below the directory where CB38 is installed.

The library file search path you have set is displayed in this field.

3.3.3.4 Default Library Path Setup Button

This button sets the directory that is set in the library file search path setup/display field as the default path to be used for CB38 when creating a new project.

When you create a new project with CB38 after setting the default path with this button, the directory you have set is used as the library file search path.

3.3.4 Library Setup Area

This area is comprised of the following five fields:

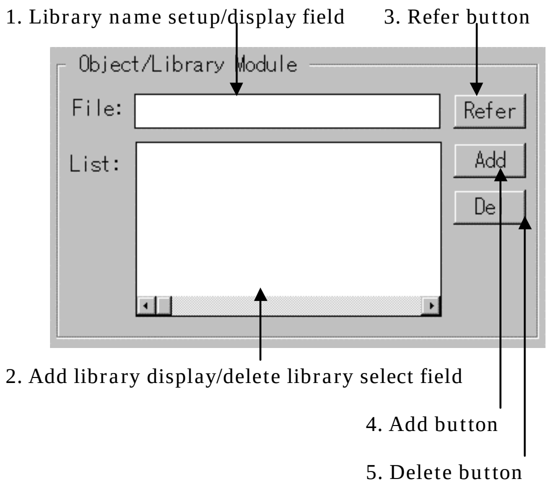

1. Library name setup/display field
3. Refer button
Object/Library Module
File:
Refer
List:
Add
De
2. Add library display/delete library select field
4. Add button
5. Delete button

Figure 26. Structure of library setup area

3.3.4.1 Library Name Setup/Display Field

In this field, set a library file name that is added to a project and is not a system library that you want to be linked when building the project.

The library file set here is added to a project as you press the "Add" button and the library file name is displayed in the add library display/delete library select field.

The library file names added to a project are listed as you press the "Add" button.

3.3.4.2 Add Library Display/Delete Library Select Field

The library file names added to a project are listed in this field.

An unnecessary library file can be deleted from a project by selecting its file name in this field by clicking on it with the mouse and pressing the "Delete" button.

3.3.4.3 Refer Button

This button allows you to add a library file to a project without having to input the file name from the keyboard.

When you press the "Refer" button, a file selection dialog box opens. The library file name you choose in this dialog box is input to the library name setup/display field. So proceed and press the "Add" button to add it to a project.

3.3.4.4 Add Button

This button adds the library file that is entered in the library name setup/display field to a project.

When you add a library file, CB38 checks to see if the file exists. If the specified library file does not exist or has already been added to a project (including system libraries), no file is added.

3.3.4.5 Delete Button

This button deletes the library file from a project that you have selected by clicking on it with the mouse in the add library display/delete library select field.

No file is deleted unless there is any library file selected.

4. Programming Language Specifications

The programming language in which programs can be written in CB38 is a subset of the C language, and is subject to the following restrictions as compared to the general C language.

- Types struct, union, and enum are nonexistent.
- Variables that involve initialization cannot be declared.
  Example:
  ```
  int a = 10;
  ```
- The static storage class is nonexistent.
- The storage class specifier that can be used is extern only.
- The types that can be used are char, int, pointer, and array only.
  Example:
  ```
  char    a;      /* 1Byte */
  int     b;      /* 4Byte */
  char*str;/* 4Byte */
  int     *p;     /* 4Byte */
  ```
- Types char and int are signed types (signed and unsigned specifiers cannot be used).
- Parameter lists cannot be written in the prototype declaration of functions.
  Example:
  ```
  int     foo(int);           /* <- Error */
  int     foo2(char *str);    /* <- Error */
  ```
- Arguments of function definitions are written in the manner similar to ANSI standards.
  Example:
  ```
  int func( int a, int b )
  {
     ...
  }
  ```
  Although parameter types are not checked when calling a function, the type of the function's return value is checked.
- Variables cannot be declared in a intra-function local block.
  Example:
  ```
  int func()
  {
      ...
      {
          int x;          /* <- Error */
      }
  }
  ```
- The preprocessor cannot expand macros accompanied by parameters. Nor can it define expressions.
  Example:
  ```
  #define  FUNC(A)        A++      /* <- Error */
  #define  EXP            label + 1 /* <- Error */
  ```
- The preprocessor pseudo-instruction #if allows only 0 or 1 to be specified in the operand.

5. Reference

5.1 Standard Functions (stdlib.lib)

The stdlib.lib provides the standard functions that can be used in custom command and custom window programs.

The prototype declaration of each function is written in stdlib.h.

Function name	Description
malloc	Allocate memory from heap area.
free	Release the area allocated by malloc.
strlen	Get the length of character string.
strcat	Concatenate character strings.
strcmp	Compare character strings.
strcpy	Copy character string.
strtoi	Convert character string into value.
gets	Input character string (from Script Window).
exit	Terminate program execution.
fopen	Open a file.
fclose	Close a file.
fseek	Move file pointer.
fgetc	Input character (from file).
fputc	Output character (to file).
fgets	Input character string (from file).
fputs	Output character string (to file).
printf	Output characters with format (to Script Window).
sprintf	Output characters with format (to memory).
fprintf	Output characters with format (to file).

5.1.1 malloc: Allocate memory from heap area

Function name:	char *malloc(int size)		
Parameter:	int	size	Number of allocated bytes
Returned value:	char *	Allocated area	
	NULL	Error	
Description:	This function allocates an area of “size” bytes from the heap area and returns the beginning address of the area. It returns NULL if there is no area that can be allocated.		

5.1.2 free: Release the area allocated by malloc() function

Function name:	int free(char *p)		
Parameter:	char	*p	Area to be released
Returned value:	0	Succeeded	
	1	Error	
Description:	This function releases the area allocated by the malloc() function.		

5.1.3 strlen: Get the length of character string

Function name:	int strlen(char *s)		
Parameter:	char	*s	Character string
Returned value:	int	Character string length of character string	
Description:	This function returns the length of s.		

5.1.4 strcat: Concatenate character strings

Function name:	char *strcat(char *s1, char *s2)		
Parameter:	char	*s1	Character string to which s2 is added
	char	*s2	Character string to be added
Returned value:	char *	Character string to which s2 is added	
Description:	This function concatenates character string s2 to the end of s1 and returns s1.		

5.1.5 strcmp: Compare character strings

Function name:	int strcmp(char *s1, char *s2)		
Parameter:	char	*s1	Character string 1
	char	*s2	Character string 2
Returned value:	Positive number		s1 > s2
	0		s1 == s2
	Negative number		s1 < s2
Description:	This function compares character string s1 and character string s2. It returns a positive number if s1 > s2 or 0 if s1 == s2 or a negative number if s1 < s2.		

5.1.6 strcpy: Copy character string

Function name:	char *strcpy(char *s1, char *s2)		
Parameter:	char	*s1	Destination
	char	*s2	Source
Returned value:	char *	Destination	
Description:	This function copies character string S2 to s1 including '¥0' and returns s1.		

5.1.7 strtoi: Convert character string into value

Function name:	int strtoi(char *str, int radix, int *value)		
Parameter:	char	*str	Character string
	int	radix	Conversion radix
	int	*value	Converted value
Returned value:	TURE	Succeeded	
	FALSE	Error	

Description: This function converts the character string specified by str into a numeric value as a value whose radix is specified by “radix”. If the conversion succeeded, the converted value is stored in *value. The values listed below can be specified for “radix”.

Value of radix	Description
0	If str begins with 0x, it is converted as a hexadecimal value; if str begins with 0, it is converted as an octal value. Otherwise, str is converted as a decimal value.
8	str is converted as an octal value.
10	str is converted as an decimal value.
16	str is converted as an hexadecimal value.

5.1.8 gets: Input character string (from Script Window)

Function name:	char *gets(char *s)		
Parameter:	char	*s	Destination in which stored
Returned value:	char *	Destination in which stored	
	NULL	Error	

Description: This function reads one line from the input area of the Script Window and stores it in s. The new-line character at the end of the line is replaced with '¥0.' The return value is stored in s. NULL is returned if an error has occurred.

5.1.9 exit: Terminate program execution

Function name:	int exit(int stat)		
Parameter:	int	stat	Program's return value
Returned value:	0	Always 0	

Description: This function terminates program execution and returns control to PD38. If stat is 0, the operation is assumed to have been processed normally. If stat is not 0, an error is assumed and the error message bearing the number that is set in macro_err is displayed in the Script Window.

5.1.10 fopen: Open a file

Function name:	int fopen(char *filename, char *attr)		
Parameter:	char	*filename	File name
	char	*attr	Open mode
Returned value:	int	File descpritor	
	NULL	Error	
Description:	This function opens the file specified by filename in the mode specified by attr. If succeeded, the return value is file descpriptor.		

5.1.11 fclose: Close a file

Function name:	int fclose(int fd)		
Parameter:	int	fd	File descriptor
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function closes the file specified by fd.		

5.1.12 fseek: Move file pointer

Function name:	int fseek(int fd, int pos, int org)		
Parameter:	int	fd	File descriptor
	int	pos	Distance the file pointer is moved
	int	org	Base point of pos
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function moves the current position in the file specified by fd at which the file is written or read. The distance of movement pos is specified as an offset from the base point org (0: Beginning of file; 1: Current position; 2: End of file).		

5.1.13 fgetc: Input character (from file)

Function name:	int fgetc(int fd)		
Parameter:	int	fd	File descriptor
Returned value:	int	read value	
	FALSE	Error	
Description:	This function reads one byte from the file pointer's current position of the file specified by fd.		

5.1.14 fputc: Output character (to file)

Function name:	int fputc(char c, int fd)		
Parameter:	char	c	Output character
	int	fd	File descriptor
Returned value:	TURE	Succeeded	
	FALSE	Error	
Description:	This function outputs one byte specified by c to the file pointer's current position of the file specified by fd.		

5.1.15 fgets: Input character string (from file)

Function name:	int fgets (char *str, int n, int fd)		
Parameter:	char	*str	Area in which to store input character string
	int	n	Maximum number of characters input
	int	fd	File descriptor
Returned value:	char *	Area in which to store input character string	
	NULL	Error	
Description:	This function reads one line from the file pointer's current position of the file specified by fd and stores it in the area specified by str.		

5.1.16 fputs: Output character string (to file)

Function name:	int fputs (char *str, int fd)		
Parameter:	char	*str	Area in which to store output character string
	int	fd	File descriptor
Returned value:	TURE	Succeeded	
	FALSE	Error	
Description:	This function outputs the character string stored in the area specified by str to the file pointer's current position of the file specified by fd.		

5.1.17 printf: Output characters with format (to Script Window)

Function name:	int printf(char *format, ...)		
Parameter:	char	*format	Format
	...	Variable parameter	
Returned value:	Positive number		Number of characters output
	Negative number		Error
Description:	This function outputs characters to the Script Window after converting them under control of “format”. The return value indicates the number of characters written to the window. A negative number is returned if an error has occurred.		

5.1.18 sprintf: Output characters with format (to memory)

Function name:	int sprintf(char *s, char *format, ...)		
Parameter:	char	*s	Output address
	char	*format	Format
	...	Variable parameter	
Returned value:	Positive number		Number of characters output
	Negative number		Error
Description:	This function outputs characters to the address specified by “s” after converting them under control of “format”. '¥0' is added at the end of output. The return value indicates the number of characters written to memory (not including '¥0'). A negative number is returned if an error has occurred.		

5.1.19 fprintf: Output characters with format (to file)

Function name:	int fprintf(int fd, char *format, ...)		
Parameter:	int	fd	File descriptor
	char	*format	Format
	...	Variable parameter	
Returned value:	Positive number		Number of characters output
	Negative number	Error	

Description: This function outputs characters to the file specified by fd after converting them under control of “format”. The return value indicates the number of characters written to the file. A negative number is returned if an error has occurred.

5.2 System Call Functions for Debugger Operation (system.lib)

The system.lib provides the system call functions that can be used in custom command and custom window programs.

The prototype declaration of each function is written in system.h.

Function name	Description
_cpu_go	Execute program in free-run mode
_cpu_gb	Execute program with break
_cpu_stop	Stop program execution
_cpu_reset	Reset the target MCU
_cpu_src_step	Execute program one source line at a time
_cpu_step	Execute program one instruction at a time
_cpu_src_over	Execute program one source line at a time including subroutines
_cpu_over	Execute program one instruction at a time including subroutines
_cpu_src_return	Return from current to calling routine one source line at a time
_cpu_return	Return from current to calling routine one instruction at a time
_cpu_wait	Wait until program execution stops
_reg_get_reg	Get register value
_reg_put_reg	Set register value
_reg_get_pc	Get program counter value
_reg_put_pc	Set program counter value
_reg_clear_cache	Clear register cache
_mem_get	Get memory value
_mem_put	Set memory value
_mem_get_endian	Get memory value with endian attached
_mem_put_endian	Set memory value with endian attached
_mem_fill	Fill memory
_mem_move	Transfer memory block
_mem_clear_cache	Clear memory cache
_break_set	Set/enable software break
_break_get	Get settings of software breaks
_break_reset	Clear software break
_break_reset_all	Clear all software breaks
_break_disable	Disable software break
_break_disable_all	Disable all software breaks
_break_enable_all	Enable all software breaks
_break_search	Get attribute of software break settings
_rram_clear	Clear RAM monitor memory
_rram_get_area	Get RAM monitor area
_rram_set_area	Set RAM monitor area
_rram_get_size	Get size of RAM monitor area
_rram_get_data	Get RAM monitor data
_info_check_run	Check execution status
_info_service	Get information on service contents
_info_cpu	Get CPU information
_info_get_map	Get map information
_info_check_map	Check mapped area

Function name	Description
_info_get_suffix	Get load file extension
_info_set_suffix	Set load file extension
_scope_set_obj	Set scope by object file name
_scope_set_addr	Set scope by address
_sym_add_sym	Enter symbols
_sym_val2sym	Get symbol for value
_sym_sym2val	Get value for symbol
_sym_add_bit	Enter bit symbols
_sym_val2bit	Get bit symbol for address and bit number
_sym_bit2val	Get address and bit number for bit symbol
_line_addr2line	Get source line for address
_line_line2addr	Get address for source line
_src_get_name	Get list of source file names
_obj_get_name	Get list of object file names
_obj_addr2obj	Get object file name by address
_func_get_name	Get list of function names
_exp_eval	Evaluate assembler expression
_com_send	Transfer sequence of bytes to emulator
_com_receive	Receive sequence of bytes from emulator
_scri_echo_on	Turn on output to script window
_scri_echo_off	Turn off output to script window
_c_exp_eval	Evaluate C-language expression
_get_shared_mem	Get shared variable
_set_shared_mem	Set shared variable
_delete_shared_mem	Delete shared variable
_get_err_msg	Get PD38's error message statement
_get_tick_count	Get elapsed time since Windows startup
_get_time	Get current system date and time
_disp_src_line	Change the contents displayed in program window
_rtt_get_range	Get RTT data range
_rtt_get_disasm	Get disassembled analysis result of RTT data
_rtt_get_bus	Get bus-mode display character string of RTT data
_rtt_check_isfetch	Check fetch cycle of RTT data
_rtt_get_data	Get RTT data
_rtt_clear_cache	Clear real-time trace (RTT) cache
_cv_get_data	Get coverage data
_cv_set_data	Set coverage data
_cv_clear_data	Clear coverage data
_cv_clear_cache	Clear coverage cache
_syscom	Execute PD38's script command
_doscom	Execute DOS command

If an error occurs, an error number written in the "Error" item is set in global variable macro_err. For details about emulator errors, refer to Section 5.2.84, "List of Emulator Errors" on page 63. For custom command programs, if FALSE is returned from the main() function, an error message corresponding to the error number that is set in macro_err is displayed in the Script Window (for PD38) or Error dialog box (for CB38).

5.2.1 _cpu_go: Execute program in free-run mode

Function name:	int _cpu_go()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function starts executing the target program from the current PC in free-run mode.	
Error:	ER_IN1_RUNNING	Already being executed
Other		Emulator error

5.2.2 _cpu_gb: Execute program with break

Function name:	int _cpu_gb()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function starts executing the target program from the current PC with breaks included.	
Error:	ER_IN1_RUNNING	Already being executed
Other		Emulator error

5.2.3 _cpu_stop: Stop program execution

Function name:	int _cpu_stop()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function stops execution of the target program.	
Error:	Emulator error	

5.2.4 _cpu_reset: Reset the target CPU

Function name:	int _cpu_reset()	
Parameter:	None	
Returned value:	TRUE Succeeded	
	FALSE Error	
Description:	This function reset the target CPU.	
Error:	ER_IN1_RUNNING	Cannot be reset because it is executing program.
	Other	Emulator error

5.2.5 _cpu_src_step: Execute program one source line at a time

Function name:	int _cpu_src_step()	
Parameter:	None	
Returned value:	TRUE Succeeded	
	FALSE Error	
Description:	This function starts executing the target program, one source line at a time, beginning with the current PC.	
Error:	ER_IN1_RUNNING	Already being executed
	ER_IN1_CANCEL	Execution suspended
	Other	Emulator error

5.2.6 _cpu_step: Execute program one instruction at a time

Function name:	int _cpu_step()	
Parameter:	None	
Returned value:	TRUE Succeeded	
	FALSE Error	
Description:	This function starts executing the target program, one instruction at a time, beginning with the current PC.	
Error:	ER_IN1_RUNNING	Already being executed
	ER_IN1_CANCEL	Execution suspended
	Other	Emulator error

5.2.7 _cpu_src_over: Execute program one source line at a time including subroutines

Function name:	int _cpu_src_over()	
Parameter:	None	
Returned value:	TRUE Succeeded	
	FALSE Error	
Description:	This function starts executing the target program, one source line at a time including subroutines, beginning with the current PC.	
Error:	ER_IN1_RUNNING	Already being executed
	ER_IN1_CANCEL	Execution suspended
	Other	Emulator error

5.2.8 _cpu_over: Execute program one instruction at a time including subroutines

Function name:	int _cpu_over()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function starts executing the target program, one instuction at a time including subroutines, beginning with the current PC.	
Error:	ER_IN1_RUNNING	Already being executed
	ER_IN1_CANCEL	Execution suspended
	Other	Emulator error

5.2.9 _cpu_src_return: Return from current to calling routine one source line at a time

Function name:	int _cpu_src_return()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function causes program execution to return from the current PC to the calling routine, one source line at a time.	
Error:	ER_IN1_RUNNING	Already being executed
	ER_IN1_CANCEL	Execution suspended
	Other	Emulator error

5.2.10 _cpu_return: Return from current to calling routine one instruction at a time

Function name:	int _cpu_return()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function causes program execution to return from the current PC to the calling routine, one instruction at a time.	
Error:	ER_IN1_RUNNING	Already being executed
	ER_IN1_CANCEL	Execution suspended
	Other	Emulator error

5.2.11 _cpu_wait: Wait until program execution stops

Function name:	int _cpu_wait()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function stops execution of a custom command or custom window program until the target program stops.	
Error:	Emulator error	

5.2.12 _reg_get_reg: Get register value

Function name:	int _reg_get_reg(int *reg, int regno)		
Parameter:	int	*reg	Register value
	int	regno	Register number
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function gets the value of the register specified by regno. In 740 Family, regno is defined as follows:

regno	Register
IN1_REG_A	A register
IN1_REG_X	X register
IN1_REG_Y	Y register
IN1_REG_S	Stack Pointer
IN1_REG_PC	Program counter
IN1_REG_F	PS register

Error: Emulator error

5.2.13 _reg_put_reg: Set register value

Function name:	int _reg_put_reg(int reg, int regno)		
Parameter:	int	reg	Register value
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function sets the value of the register specified by regno. The definition of regno here is the same as for the _reg_get_reg() function.

Error:	ER_IN1_DATA_OUTRANGE	Data range is invalid.
	Other	Emulator error

5.2.14 _reg_get_pc: Get program counter value

Function name:	int _reg_get_pc(int *pc)		
Parameter:	int	*pc	Program counter (PG register + PC register) value
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function gets the program counter (PG register +PC register) value.

Error: Emulator error

5.2.15 _reg_put_pc: Set program counter value

Function name:	int _reg_put_pc(int pc)		
Parameter:	int	pc	Program counter (PG register + PC register) value
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function sets a program counter (PG register + PC register) value.		
Error:	ER_IN1_ADDR_OUTRANGE		Address range is invalid
	Other		Emulator error

5.2.16 _reg_clear_cache: Clear register cache

Function name:	int _reg_clear_cache()	
Parameter:	None	
Returned value:	TRUE	Return value is always TRUE.
Description:	This function clears the register cache.	

5.2.17 _mem_get: Get memory value

Function name:	int _mem_get(int addr, int size, char *data)		
Parameter:	int	addr	Address
	int	size	Number of bytes obtained
	char	*data	Location where obtained data is stored
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function stores "size" bytes of data from addr into "data".		
Error:	ER_IN1_ADDR_OUTRANGE		Address range is invalid.
	ER_IN1_RUNNING		Cannot be obtained because program is executing.
	Other		Emulator error

5.2.18 _mem_put: Set memory value

Function name:	int _mem_put(int addr, int size, char *data)		
Parameter:	int	addr	Address
	int	size	Number of bytes set
	char	*data	Set data
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function sets data data from addr into "size" bytes of memory.		
Error:	ER_IN1_ADDR_OUTRANGE		Address range is invalid.
	ER_IN1_RUNNING		Cannot be set because program is executing.
	Other		Emulator error

5.2.19 _mem_get_endian: Get memory value with endian attached

Function name: int _mem_get_endian(int addr, int num, int size, int *data)

Parameter:	int	addr	Address
	int	num	Number of data entries
	int	size	Size of one data entry
	int	*data	Location where obtained data is stored
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function stores num entries of data in data size of “size” bytes from addr into data[] according to the CPU endian. Numerals 1 to 4 can be specified for “size”.

Error:	ER_IN1_ADDR_OUTRANGE	Address range is invalid.
	ER_IN1_DATA_RANGE	“size” is not 1 to 4.
	ER_IN1_RUNNING	Cannot be obtained because program is executing.
	Other	Emulator error

5.2.20 _mem_put_endian: Set memory value with endian attached

Function name: int _mem_put_endian(int addr, int num, int size, int *data)

Parameter:	int	addr	Address
	int	num	Number of data entries
	int	size	Size of one data entry
	int	*data	Set data
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function sets num entries of data in data size of “size” bytes from data[] into memory locations beginning with addr according to the CPU endian

Error:	ER_IN1_ADDR_OUTRANGE	Address range is invalid.
	ER_IN1_DATA_RANGE	“size” is not 1 to 4.
	ER_IN1_RUNNING	Cannot be set because program is executing.
	Other	Emulator error

5.2.21 _mem_fill: Fill memory

Function name:	int _mem_fill(int start, int end, int data, int size)		
Parameter:	int	start	Start address
	int	end	End address
	int	data	Filled data
	int	size	Size of one data entry
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function fills a memory area from “start” to “end” with data data in data size of “size” bytes.

Error:	ER_IN1_ADDR_OUTRANGE	Address range is invalid.
	ER_IN1_DATA_RANGE	“size” is not 1 to 4.
	ER_IN1_RUNNING	Cannot be filled because program is executing.
	Other	Emulator error

5.2.22 _mem_move: Transfer memory block

Function name:	int _mem_move(int start, int end, int top)		
Parameter:	int	start	Start address
	int	end	End address
	int	top	Beginning address at destination of transfer
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function transfers data at addresses from start to end to an area beginning with top.

Error:	ER_IN1_ADDR_OUTRANGE	Address range is invalid.
	ER_IN1_RUNNING	Cannot be tramsferd because program is executing.
	Other	Emulator error

5.2.23 _mem_clear_cache: Clear memory cache

Function name:	int _mem_clear_cache()	
Parameter:	None	
Returned value:	TRUE	Return value is always TRUE.

Description: This function clears the cache buffer for a module that gets cached memory.

5.2.24 _break_set: Set/enable software break

Function name: int _break_set(int addr)

Parameter: int addr Set address

Returned value: TRUE Succeeded
FALSE Error

Description: This function sets a software breakpoint at “addr”. This function also is used to re-enable a breakpoint that has been disabled by _break_disable() or _break_disable_all()

Error:

ER_IN1_ADDR_OUTRANGE	Address range is invalid.
ER_IN1_RUNNING	Cannot be set because program is executing.
ER_IN1_BP_FULL	Breakpoints are full.
Other	Emulator error

5.2.25 _break_get: Get settings of software breaks

Function name: int _break_get(int *addr, int *attr, int mode)

Parameter:
int *addr Address
int *attr Setup attribute
int mode Search start mode
IN1_FIRST : First breakpoint
IN1_NEXT : Second and following breakpoints

Returned value: TRUE Succeeded
FALSE Error

Description: This function stores a breakpoint address in *addr. One of the breakpoint setup attributes shown below is stored in *attr.

IN1_ENABLE_SBRK	Enabled
IN1_DISABLE_SBRK	Disabled

Error:

ER_IN1_RUNNING	Cannot be obtained because program is executing.
ER_IN1_BP_NOTFOUND	No breakpoint can be found.
Other	Emulator error

5.2.26 _break_reset: Clear software break

Function name: int _break_reset(int addr)

Parameter: int addr Address

Returned value: TRUE Succeeded
FALSE Error

Description: This function clears a breakpoint at addr.

Error:

ER_IN1_ADDR_OUTRANGE	Address range is invalid.
ER_IN1_RUNNING	Cannot be cleard because program is executing.
ER_IN1_BP_NOTFOUND	No breakpoint can be found.
Other	Emulator error

5.2.27 _break_reset_all: Clear all software breaks

Function name:	int _break_reset_all()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function clears all breakpoints.	
Error:	ER_IN1_RUNNING	Cannot be cleard because program is executing.
	Other	Emulator error

5.2.28 _break_disable: Disable software break

Function name:	int _break_disable(int addr)	
Parameter:	int addr	Address
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function disables a breakpoint at addr.	
Error:	ER_IN1_ADDR_OUTRANGE	Address range is invalid.
	ER_IN1_RUNNING	Cannot be disabled because program is executing.
	ER_IN1_BP_NOTFOUND	No breakpoint can be found.
	Other	Emulator error

5.2.29 _break_disable_all: Disable all software breaks

Function name:	int _break_disable_all()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function disables all breakpoints set.	
Error:	ER_IN1_RUNNING	Cannot be disabled because program is executing.
	Other	Emulator error

5.2.30 _break_enable_all: Enable all software breaks

Function name:	int _break_enable_all()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function enables all breakpoints set.	
Error:	ER_IN1_RUNNING	Cannot be enabled because program is executing.
	Other	Emulator error

5.2.31 _break_search: Get attribute of software break settings

Function name:	int _break_search(int addr, int *attr)		
Parameter:	int	addr	Address
	int	*attr	Setup attribute
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function gets the setup attribute of a breakpoint at addr. One of the following breakpoint setup attributes is stored in *attr.

IN1_ENABLE_SBRK	Enabled
IN1_DISABLE_SBRK	Disabled

Error:	ER_IN1_RUNNING	Cannot be obtained because program is executing.
	ER_IN1_BP_NOTFOUND	No breakpoint can be found.
	Other	Emulator error

5.2.32 _rram_clear: Clear RAM monitor memory

Function name:	int _rram_clear()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error

Description: This function initializes access states of the RAM monitor memory.

Error:	ER_IN1_RUNNING	Cannot be cleard because program is executing.
	Other	Emulator error

5.2.33 _rram_get_area: Get RAM monitor area

Function name:	int _rram_get_area(int *addr)		
Parameter:	int	*addr	Beginning address
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function stores the beginning address of the RAM monitor memory in *addr.

Error: Emulator error

5.2.34 _rram_set_area: Set RAM monitor area

Function name:	int _rram_set_area(int addr)		
Parameter:	int	addr	Beginning address
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function sets the beginning address of the RAM monitor area at addr.

Error:	ER_IN1_ADDR_OUTRANGE	Address range is invalid.
	ER_IN1_RUNNING	Cannot be set because program is executing.
	Other	Emulator error

5.2.35 _rram_get_size: Get size of RAM monitor area

Function name:	int _rram_get_size(int *size)		
Parameter:	int	*size	size of RAM monitor area
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function sets the size of the RAM monitor area in *size.		

5.2.36 _rram_get_data: Get RAM monitor data

Function name:	int _rram_get_data(int addr, int size, char *data, char *attr)		
Parameter:	int	addr	Beginning address
	int	size	Number of bytes
	char	*data	Data
	char	*attr	Access state
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function gets “size” bytes of data (*data) beginning with addr and access state (*attr) from the RAM monitor. One of the access states shown below is stored in *attr.

IN1_RRAM_READ	Read
IN1_RRAM_WRITE	Write
IN1_RRAM_NONE	No access

Error:	ER_IN1_ADDR_OUTRANGE	Address range is invalid.
	Other	Emulator error

5.2.37 _info_check_run: Check execution status

Function name:	int _info_check_run(int *status)		
Parameter:	int	*status	Execution state
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function stores the execution state of the target program in *status. One of the following execution status is stored in *status.

IN1_RUN_CPU	Under execution
IN1_STOP_CPU	Idle

Error: Emulator error

5.2.38 _info_service: Get information on service contents

Function name: int _info_service(int flag, int *status)

Parameter: int flag Service content

int *status Availability of support

TRUE	Supported
FALSE	Not supported

Returned value: TRUE Return value is always TRUE.

Description: This function gets information on service contents supported by PD38. For flag, specify one of the following service contents.

IN1_SUPPORT_BITSYM	Support for bit symbol
IN1_SUPPORT_C	Support for C-language debugging
IN1_SUPPORT_RAMMONITOR	Support for real-time RAM monitor function
IN1_SUPPORT_RTT	Support for real-time trace
IN1_SUPPORT_CV	Support for coverage measurement
IN1_SUPPORT_PROTCT	Support for protected break
IN1_SUPPORT_EVENT	Support for hardware event

5.2.39 _info_cpu: Get CPU information

Function name: int _info_cpu(int flag, int *status)

Parameter: int flag Content of information

int *status CPU information

IN1_BIG_ENDIAN	Big endian
IN1_LITTLE_ENDIAN	Little endian
Other	Value corresponding to flag

Returned value: TRUE Return value is always TRUE.

Description: This function gets information on the target CPU. For “flag”, specify one of the following information.

IN1_ADDRSIZE	Number of bytes required for storing address value
IN1_MAXADDR	Maximum value of address
IN1_ADDRCOLM	Number of digits with which address values are displayed in hexadecimal
IN1_ENDIAN	Endian of the target CPU
IN1_WORD_SIZE	Length in bytes of word
IN1_MAXDATA	Maximum value of data
IN1_MAXSTACK	Maximum value of stack
IN1_MAX_OBJ	Maximum length in bytes of one instruction

5.2.40 _info_get_map: Get map information

Function name:	int _info_get_map(int *start, int *end, int mode)		
Parameter:	int	*start	Start address
	int	*end	End address
	int	mode	Search start mode IN1_FIRST : First map IN1_NEXT : Second and following maps
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function gets map information. The start and the end addresses of a mapped area are stored in *start and *end, respectively.

Error: IN1_MAP_END No more map

5.2.41 _info_check_map: Check mapped area

Function name: int _info_check_map(int start, int end, int *status, int *erradr)

Parameter:	int	start	Start address
	int	end	End address
	int	*status	Check result
	int	*erraddr	Error address
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function checks to see if the address range from “start” to end is a mapped area. If the address range from “start” to “end” entirely is a mapped area, TRUE is stored in *status. If the address range from “start” to “end” contains any unmapped area, FALSE is stored in *status and the address of the first unmapped area found by searching from start is stored in erraddr.

Error:	ER_IN1_ADDR_OUTRANGE	Address range is invalid.
	Other	Emulator error

5.2.42 _info_get_suffix: Get load file extension

Function name:	int _info_get_suffix(char *suffix, int mode)		
Parameter:	char	*suffix	Obtained extension
	int	mode	Mode
Returned value:	TRUE	Return value is always TRUE.	

Description: This function gets a file suffix that is added in a file selection dialog box when downloading the target program in the mode specified by “mode”. For mode, specify one of the following attributes.

IN1_LOAD	Symbol and program files
IN1_SYM	Symbol file
IN1_ROM	Program file

5.2.43 _info_set_suffix: Set load file extension

Function name:	int _info_set_suffix(char *suffix, int mode)
Parameter:	char *suffix Extension to be set int mode Mode
Returned value:	TRUE Return value is always TRUE.
Description:	This function sets a file suffix that is added in a file selection dialog box when downloading the target program in the mode specified by “mode”. For “mode”, specify one of the following attributes.

IN1_LOAD	Symbol and program files
IN1_SYM	Symbol file
IN1_ROM	Program file

5.2.44 _info_ispc4700h: Identify connected emulator

Function name:	int _info_ispc4700h()
Parameter:	None
Returned value:	TRUE Connected emulator is PC4700H or PC4701HS FALSE Connected emulator is not the above.
Description:	This function gets information on the type of emulator that is connected to the emulation system. The coverage measurement and real-time trace functions can only be used with the PC4700H and PC4701HS. Therefore, when using the functions that begin with _cv_ or _rtt_, you need to check that the connected emulator is the PC4700H or PC4701HS.

5.2.45 _scope_set_obj: Set scope by object file name

Function name:	int _scope_set_obj(char *name)
Parameter:	char *name Object file name
Returned value:	TRUE Succeeded FALSE Error
Description:	This function sets the current scope by an object file name.
Error:	ER_SCOPE_NOTFOUND No scope is found that corresponds to the specified object file name.

5.2.46 _scope_set_addr: Set scope by address

Function name:	int _scope_set_addr(int addr)
Parameter:	int addr Address
Returned value:	TRUE Succeeded FALSE Error
Description:	This function sets the current scope by an address.
Error:	ER_IN1_ADDR_OUTRANGE Address range is invalid.

5.2.47 _sym_add_sym: Enter symbols

Function name:	int _sym_add_sym(int mode, char *name, int value)		
Parameter:	int	mode	Search mode
	char	*name	Symbol
	int	value	Value
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function enters the symbol (or label) “name” as a global symbol (or label). For”mode”, specify one of the following types.

LOAD_SYMBOL	Symbol first
LOAD_LABEL	Label first

Error:

ER_LOAD_ILLEGAL_CHAR	Character string contains a character that cannot be used for a symbol or label.
ER_LOAD_MULTIDEFINE	A global symbol (or label) of the same name already exists.

5.2.48 _sym_val2sym: Get symbol for value

Function name:	int _sym_val2sym(int mode, int value, char *symbol)		
Parameter:	int	mode	Search mode
	int	value	Value
	char	*symbol	Area in which symbol is stored
Returned value:	TRUE	Succeeded	
	FALSE	Corresponding symbol could not be found.	

Description: This function searches for a symbol character string that corresponds to a value “value” and stores it in “symbol”. For “mode”, specify one of the priorities of search shown below.

LOAD_SYMBOL	Symbol first
LOAD_LABEL	Label first

The table below shows the priorities of search in each mode.

	Searched symbol first		Searched label first
1	Local symbol (within scope)	1	Local label (within scope)
2	Global symbol	2	Global label
3	Local label (within scope)	3	Local symbol (within scope)
4	Global label	4	Global symbol
5	Local symbol (outside scope)	5	Local label (outside scope)
6	Local label (outside scope)	6	Local symbol (outside scope)

5.2.49 _sym_sym2val: Get value for symbol

Function name:	int _sym_sym2val(int mode, char *symbol, int *value)		
Parameter:	int	mode	Search mode
	char	*symbol	Symbol
	int	*value	Value
Returned value:	TRUE	Succeeded	
	FALSE	Symbol could not be found.	
Description:	This function searches for a value that corresponds to the symbol character string “symbol” and stores it in “*value”. The specified mode here is the same as for _sym_val2sym().		
Error:	ER_LOAD_SYMBOL_NOTFOUND		Symbol cannot be found.

5.2.50 _sym_add_bit: Enter Bit symbols

Function name:	int _sym_add_bit(char *bitsym, int addr, int bit)		
Parameter:	char	*bitsym	Bit Symbol
	int	addr	Address
	int	bit	Bit number
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function enters the bit symbol “bitsym” as a global bit symbol.		
Error:	ER_LOAD_ILLEGAL_CHAR		Character string contains a character that cannot be used for a bit symbol.
	ER_LOAD_MULTIDEFINE		A global bit symbol of the same name already exists.
	ER_LOAD_ADDR_OUTRANGE		Address range is invalid.
	ER_LOAD_BIT_OUTRANGE		Bit number range is invalid.

5.2.51 _sym_val2bit: Get bit symbol for address and bit number

Function name:	int _sym_val2bit(int addr, int bit, char *bitsym)		
Parameter:	int	addr	Address
	int	bit	Bit number
	char	*bitsym	Area in which bit symbol is stored
Returned value:	TRUE	Succeeded	
	FALSE	Corresponding bit symbol could not be found.	
Description:	This function searches for a bit symbol character string that corresponds to an “address” and a “bit” and stores it in “*bitsym”.		

5.2.52 _sym_bit2val: Get address and bit number for bit symbol

Function name:	int _sym_sym2val(char *bitsym, int *addr, int *bit)		
Parameter:	char	*bitsym	Bit symbol
	int	*addr	Address
	int	*bit	Bit number
Returned value:	TRUE	Succeeded	
	FALSE	Bit symbol could not be found.	
Description:	This function searches for an address and a bit number that corresponds to the bit symbol character string “bitsym” and stores it in “*addr” and “*bit”.		
Error:	ER_LOAD_SYMBOL_NOTFOUND	Bit symbol cannot be found.	

5.2.53 _line_addr2line: Get source line for address

Function name:	int _sym_addr2line(int addr, int *line, char *filename)		
Parameter:	int	addr	Address
	int	*line	Line number
	char	*filename	Area where file name is stored
Returned value:	TRUE	Succeeded	
	FALSE	Source line information cannot be found.	
Description:	This function gets the line number (*line) that corresponds to the address addr and its file name (filename).		
Error:	ER_LOAD_FILE_NOTFOUND	File cannot be found.	
	ER_LOAD_SRCINF_NOTFOUND	Source information cannot be found.	

5.2.54 _line_line2addr: Get address for source line

Function name:	int _sym_line2addr(char *filename, int line, int *addr)		
Parameter:	char	*filename	File name
	int	line	Line number
	int	*addr	Address
Returned value:	TRUE	Succeeded	
	FALSE	Source line information cannot be found.	
Description:	This function gets the address (*addr) that corresponds to the line (line) in the file (filename).		
Error:	ER_LOAD_LINE_NOTFOUND	Line information cannot be found.	

5.2.55 _src_get_name: Get list of source file names

Function name:	int _src_get_name(char *objname, char *srcname, int mode)		
Parameter:	char	*objname	Object file name
	char	*srcname	Area where source file name is stored
	int	mode	Search start mode
		LOAD_FIRST : First source file name	
		LOAD_NEXT : Second and following source file names	
Returned value:	TRUE	Succeeded	
	FALSE	Source file name cannot be found.	
Description:	This function gets a list of source file names in the object file “objname”. If NULL is specified for object, a list of source file names in all object files is obtained.		

5.2.56 _obj_get_name: Get list of object file names

Function name:	int _obj_get_name(char *objname, int mode)		
Parameter:	char	*objname	Area where object file name is stored
	int	mode	Search start mode
		LOAD_FIRST : First source file name	
		LOAD_NEXT : Second and following source file names	
Returned value:	TRUE	Succeeded	
	FALSE	Object file name cannot be found.	
Description:	This function gets a list of object file names.		

5.2.57 _obj_addr2obj: Get object file name by address

Function name:	int _obj_addr2obj(int addr, char *objname)		
Parameter:	int	addr	Address
	char	*objname	Area where object file name is stored
Returned value:	TRUE	Succeeded	
	FALSE	Corresponding object file name cannot be found.	
Description:	This function gets the object file name objname that contains the address addr.		

5.2.58 _func_get_name: Get list of function names

Function name:	int _func_get_name(char *objname, char *funcname, int mode)		
Parameter:	char	*objname	Object file name
	char	*funcname	Area where function name is stored
	int	mode	Search start mode
		LOAD_FIRST : First source file name	
		LOAD_NEXT : Second and following source file names	
Returned value:	TRUE	Succeeded	
	FALSE	Function name cannot be found.	
Description:	This function gets a list of function names in the object file “objname”. If NULL is specified for “objname”, a list of function names in all object files is obtained.		

5.2.59 _exp_eval: Evaluate assembler expression

Function name:	int _exp_eval(char *exp, int radix, int mode, int *value)		
Parameter:	char	*exp	Assembler expression
	int	radix	Radix
	int	mode	Priorities in which symbols (labels) are evaluated
	int	*value	Area where analysis result is stored
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function evaluates the assembler expression (exp) and stores the evaluation result in *value. For radix, specify one of the radices of constants shown below.

EXP_DEC	Decimal
EXP_HEX	Hexadecimal
EXP_DEFAULT	Value set by RADIX command is used

For "mode", specify one of the priorities of symbol (label) evaluation shown below.

EXP_SYMBOL	Symbol first
EXP_LABEL	Label first

Error:

ER_EXP_SYNTAX	Syntax error
ER_EXP_ZERO	Divide-by-0 error
ER_EXP_LPAR	Left parenthesis missing
ER_EXP_SIZE	Incorrect size specifier
ER_EXP_STRING	Character string not terminated
ER_EXP_LINE	Incorrect line number specified
ER_EXP_VALUE	Incorrect constant value specified
ER_EXP_UNDEF_SYMBOL	Symbol not defined
ER_EXP_PREFIX	Incorrect radix of constant specified
ER_EXP_OVER	Constant value out of range
ER_EXP_UNDEF_MACRO	Macro constant not defined
ER_EXP_ILLEGAL_MACRO	Register name used for macro variable name
ER_EXP_OUTOF_MACRO	Limit number of macro constants exceeded

5.2.60 _com_send: Transfer sequence of bytes to emulator

Function name:	int _com_send(char *data, int size)		
Parameter:	char	*data	Sequence of bytes
	int	size	Number of bytes transferred
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function transfers “size” bytes of a byte sequence specified by data.		
Error:	Emulator error		

5.2.61 _com_receive: Receive sequence of bytes from emulator

Function name:	int _com_receive(char *data, int size)		
Parameter:	char	*data	Sequence of bytes
	int	size	Number of bytes received
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function receives “size” bytes of data and stores them in data. If size bytes of data cannot be received, FALSE is returned.		
Error	Emulator error		

5.2.62 _scri_echo_on: Turn on output to script window

Function name:	int _scri_echo_on()	
Parameter:	None	
Returned value:	TRUE	Return value is always TRUE.
Description:	This function turns output to the Script Window on. By default, the Script Window is enabled for output.	

5.2.63 _scri_echo_off: Turn off output to script window

Function name:	int _scri_echo_off()	
Parameter:	None	
Returned value:	TRUE	Return value is always TRUE.
Description:	This function turns output to the Script Window off.	

5.2.64 _c_exp_eval: Evaluate C-language expression

Function name: int _c_exp_eval(char *exp, int *value1, int *value2, char *type, char *str, char *misc)

Parameter:

char	*exp	C-language expression
int	*value1	Analysis result 1
int	*value2	Analysis result 2
char	*type	Character string showing type of analysis result
char	*str	Character string showing analysis result
char	*misc	Character string showing added information of analysis result

Returned value:

TRUE	Succeeded
FALSE	Error

Description: This function analyzes the C-language expression specified by exp in the current scope. The analysis result is a 64-bit value, with the 32 low-order bits stored in *value1 and the 32 high-order bits stored in *value2. The type name of the analysis result is stored in “type” and the analysis result is stored in str after being converted into a character string. If the analysis result is not one that indicates an ordinary value such as a function, addition information is stored in misc. The information stored in “type”, str, and misc can be output for display using the printf() function in the same way as possible with a script command "print".

Error:

ER_CEXP_NOT_FOUND	Symbol cannot be found.
ER_CEXP_SYNTAX_ERROR	Syntax error.
ER_CEXP_NO_SCOPE	Scope cannot be found.
ER_CEXP_NO_SYMBOL	Symbol cannot be found.
ER_CEXP_NO_FUNC	Function cannot be found.
ER_CEXP_RIGHT_WRONG	Right-side expression is inappropriate.
ER_CEXP_DEF_TYPE	Copying different type of structure (union) is inhibited.
ER_CEXP_CANT_ASSIGN	Cannot be substituted.
ER_CEXP_NO_TYPE	Type cannot be found.
ER_CEXP_CANT_FLOAT	Floating-point operation is not supported.
ER_CEXP_CANT_P_TO_P	Specified operation cannot be performed between pointer types.
ER_CEXP_CANT_SUB_P	Specified operation cannot be performed on pointer type.
ER_CEXP_CANT_P	Subtraction by pointer variable cannot be performed.
ER_CEXP_0_DIV	Divide-by-0 is attempted.
ER_CEXP_UNKNOWN_OP	Invalid operator is used.
ER_CEXP_BROKEN_TYPE	Type information is broken.
ER_CEXP_LEFT_POINT	Left-side value must be a pointer variable.

ER_CEXP_LEFT_STRUCT	Left-side value must be a structure (union) type.
ER_CEXP_NO_MEMBER	Member cannot be found.
ER_CEXP_LEFT_STRUCT_REF	Left-side value must be a refarence of structure (union) type.
ER_CEXP_LEFT_WRONG	Left-side value is inappropriate.
ER_CEXP_VAL_NUM	Operand must be a numeric value.
ER_CEXP_CANT_MIN	Specified operand cannot be sign-inverted.
ER_CEXP_CANT_REF	Address value cannot be obtained.
ER_CEXP_LEFT_ARRAY	Array variable is inappropriate.
ER_CEXP_RIGHT_NUM	Element numbers of the array is inappropriate.
ER_CEXP_NOT_POINT	Operand is not an address.
ER_CEXP_CANT_CAST_REG	Cast operation on variables is not supported.
ER_CEXP_CANT_CAST	Specified type to be cast is inappropriate.
ER_CEXP_CAST_NOT_SUPPORT	Cast operation on specified type is not supported.

5.2.65 _get_shared_mem: Get shared variable

Function name:	int _get_shared_mem(char *name, char *value)		
Parameter:	char	*name	Name of shared variable
	char	*value	Value of shared variable
Returned value:	TRUE	Succeeded	
	FALSE	Shared variable cannot be found.	

Description: This function searches for the shared variable specified by "name" and stores its value (character string) in value. A shared variable means a variable that can be used in common in all custom command and custom window programs. The name and the value of a shared variable are a character string and can be used in a similar manner as the environment variables found in several operation systems. The name and the value of a shared variable can be used in up to 63 bytes.

5.2.66 _set_shared_mem: Set shared variable

Function name:	int _set_shared_mem(char *name, char *value)		
Parameter:	char	*name	Name of shared variable
	char	*value	Value of shared variable
Returned value:	TRUE	Return value is always TRUE.	

Description: This function sets the shared variable specified by "name" in the value specified by "value". If a value is set for the shared variable that has already been set, the previously set value is changed to the value specified by value. If the shared variable is not defined, a new variable area is allocated.

5.2.67 _delete_shared_mem: Delete shared variable

Function name:	int _delete_shared_mem(char *name)		
Parameter:	char	*name	Name of shared variable
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function deletes the shared variable that is specified by “name”. If the shared variable is not defined, nothing is performed.		

5.2.68 _get_err_msg: Get PD38's error message statement

Function name:	int _get_err_msg(int err_no, char *msg)		
Parameter:	int	err_no	Error number
	char	*msg	Error message statement
Returned value:	TRUE	Succeeded	
	FALSE	Error Error message statement corresponding to error number cannot be found.	
Description:	This function gets PD38's error message statement that corresponds to the error number specified by err_no.		

5.2.69 _get_tick_count: Get elapsed time since Windows startup

Function name:	int _get_tick_count()
Parameter:	None
Returned value:	Elapsed time since Windows startup (in ms)
Description:	This function gets an elapsed time in ms since Windows has started up.

5.2.70 _get_time: Get current system date and time

Function name: int _get_time(int *year, int *month, int *dayOfWeek,
int *day, int *hour, int *minute,
int *secont, int *milliseconds)

Parameter:	int	*year	Location where current year is stored
	int	*month	Location where current month (1-12) is stored
	int	*dayOfWeek	Location where current day of the week (e.g., Sunday = 0) is stored
	int	*day	Location where current day (1-31) is stored
	int	*hour	Location where current time in hours (1-24) is stored
	int	*minute	Location where current time in minutes (0-59) is stored
	int	*second	Location where current time in seconds (0-59) is stored
	int	*milliseconds	Location where current time in milliseconds (0-999) is stored

Returned value: TRUE Return value is always TRUE.

Description: This function gets the current date and time of the system and stores them in the locations specified by each parameter. If NULL is specified for a parameter, the information corresponding to that parameter is not stored.

5.2.71 _disp_src_line: Change the contents displayed in program window

Function name: int disp_src_line(int lineno, char *filename, int addr)

Parameter:	int	lineno	Line number
	char	*filename	File name
	int	addr	Address

Returned value:	TRUE	Succeeded
	FALSE	Error

Description: This function updates the contents displayed in PD38's program window. The selected line of the selected (specified by lineno and filename) is displayed in the program window in the source mode. If selected line of the selected source file cannot be displayed, the file is displayed in the disassemble mode beginning with the address specified by “addr”.

5.2.72 _rtt_get_range: Get RTT data range

Function name:	int _rtt_get_range(int *s_cycle, int *e_cycle)		
Parameter:	int	*s_cycle	Start cycle
	int	*e_cycle	End cycle
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function gets a range of trace data from start to end cycles data that can be referenced.		
Error:	ER_IN2_NONE_RTT		Referencible trace data cannot be found.
	Other		Emulator error

5.2.73 _rtt_get_disasm: Get disassembled analysis result of RTT data

Function name:	int _rtt_get_disasm(int *cycle, int *next_cycle, char *buffer, int *count)		
Parameter:	int	*cycle	Analysis start/analysis result cycle
	int	*next_cycle	Next fetch cycle
	char	*buffer	Area where character string representing analysis result is stored
	int	*count	Number of instructions fetched (always 1)
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function searches for a fetch cycle beginning with the cycle specified by *cycle and, when found, stores the fetch cycle in *cycle and disassembles the instruction before storing it in buffer. The next fetch cycle is stored in *next_cycle..		
Error:	ER_IN2_NONE_RTT		Referencible trace data cannot be found.
	ER_IN2_CYCLE_OUTRANGE		Specified cycle value is out of range.
	Other		Emulator error

5.2.74 _rtt_get_bus: Get bus-mode display character string of RTT data

Function name:	int _rtt_get_bus(int cycle, char *buffer)		
Parameter:	int	cycle	Cycle
	char	*buffer	Area where character string is stored
Returned value:	TRUE	Succeeded	
FALSE	Error		

Description: This function converts the trace data in the cycle specified by cycle into a character string for display purpose and stores the result in buffer.

Error:	ER_IN2_NONE_RTT	Referencible trace data cannot be found.
	ER_IN2_CYCLE_OUTRANGE	Specified cycle value is out of range.
	Other	Emulator error

5.2.75 _rtt_check_isfetch: Check fetch cycle of RTT data

Function name:	int _rtt_check_isfetch(int cycle, int *addr1, int *addr2, int *count)		
Parameter:	int	cycle	Check cycle
	int	*addr1	Fetch address 1
	int	*addr2	Fetch address 2(Not Used)
	int	*count	Number of instructions fetched
Return value:	TRUE	Succeeded	
FALSE	Error		

Description: This function checks the cycle specified by cycle to see if it is a fetch cycle. If it is not a fetch cycle, 0 is stored in *count. If it is a fetch cycle, 1 is stored in *count.

Error:	ER_IN2_NONE_RTT	Referencible trace data cannot be found.
	ER_IN2_CYCLE_OUTRANGE	Specified cycle value is out of range.
	Other	Emulator error

5.2.76 _rtt_get_data: Get RTT data

Function name:	_rtt_get_data(int cycle, int *addr, int *data, int *state, int *ext, int *dn, int *h, int *m, int *s, int *ms, int *us)		
Parameter:	int	cycle	Cycle
	int	*addr	Value of address bus
	int	*data	Value of data bus
	int	*state	Value of CPU status signal
	int	*ext	Value of external trigger signal
	int	*dn	Number of bytes occupying queue buffer (always 0)
	int	*h	Hours
	int	*m	Minutes
	int	*s	Seconds
	int	*ms	Milliseconds
	int	*us	Microseconds
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function gets the RTT data in the cycle specified by cycle. The value of the address bus and that of the data bus are stored in *addr and *data, respectively. The CPU status signals shown below are stored in *state.

CPU status, low-order

DMA	0	SYNC	0	SYNC	ONW*	WR*	RD*

(The asterisk "*" in the above table denotes that the signal is active low.)

The external trigger input value is stored in *ext. The time in hours, minutes, seconds, milliseconds, and microseconds since program execution started are stored in *h, *m, *s, *ms, and *us, respectively.

Error:	ER_IN2_NONE_RTT	Referencible trace data cannot be found.
	ER_IN2_CYCLE_OUTRANGE	Specified cycle value is out of range.
	Other	Emulator error

5.2.77 _rtt_clear_cache: Clear real-time trace (RTT) cache

Function name:	int _rtt_clear_cache()	
Parameter:	None	
Returned value:	TRUE	Return value is always TRUE.
Description:	This function clears the real-time trace (RTT) cache.	

5.2.78 _cv_get_data: Get coverage data

Function name:	int _cv_get_data(int saddr, int eaddr, int *rsaddr, int *readdr, char *data)		
Parameter:	int	saddr	Start address of data to be obtained
	int	eaddr	End address of data to be obtained
	int	*rsaddr	Start address of data actually obtained
	int	*readdr	End address of data actually obtained
	char	*data	Coverage data obtained
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function stores the coverage data that includes an address range specified by s_addr and e_addr in the area specified by "data". However, since data for 8 bytes of addresses from each 8-byte alignment is stored in one byte of "data", it can happen that a greater range of data than addresses specified by s_addr and e_addr actually is stored. For example, if addresses from 3h to 19h are specified, data at addresses from 0h to 1Fh actually are stored. The start and end addresses of the actually obtained data are stored in *rs_addr and *re_addr, respectively. Note that the values stored in *rs_addr and *re_addr can be obtained by calculation using the formula below.

*rs_addr = s_addr / 8 * 8

*re_addr = e_addr / 8 * 8 + 7

For "data", specify an array greater than e_addr - s_addr / 8 + 1. The format of the coverage data stored in one byte of "data" is as follows:

(Upper row: Bit offset; Lower row: address offset)

7	6	5	4	3	2	1	0
+7	+6	+5	+4	+3	+2	+1	+0

For example, if s_addr is 0x400, the coverage results at the addresses offset by the amount corresponding to each bit are stored in "data[0]" as shown below.

(Upper row: Bit offset; Lower row: Address)

7	6	5	4	3	2	1	0
407	406	405	404	403	402	401	400

Consequently, if memory is accessed every other byte beginning with s_addr, coverage data is stored as shown below.

(Upper row: Bit offset; Lower row: Coverage measurement result)

7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1

The data stored in data[0] is 01010101B, i.e., 0x55.

Error:	ER_IN2_ADDR_OUTRANGE	Specified address is out of range.
	ER_IN2_RUNNING	Cannot be obtained because program is executing.
	Other	Emulator error

5.2.79 _cv_set_data: Set coverage data

Function name:	int _cv_set_data(int s_addr, int e_addr, char *data)		
Parameter:	int	s_addr	Set start address
	int	e_addr	Set end address
	char	*data	Set coverage data
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function sets the coverage data stored in the area specified by “data” in a range of addresses specified by s_addr and e_addr. However, since the coverage data stored in one byte of “data” is for 8 bytes of addresses, specify values for s_addr and e_addr in increments of 8 bytes. The format of “data” is the same as for the _cv_get_data() function described above.		

Error:	ER_IN2_ADDR_OUTRANGE	Specified address is out of range.
	ER_IN2_RUNNING	Cannot be set because program is executing.
	Other	Emulator error

5.2.80 _cv_clear_data: Clear coverage data

Function name:	int _cv_clear_data()	
Parameter:	None	
Returned value:	TRUE	Succeeded
	FALSE	Error
Description:	This function clears coverage data.	
Error:	ER_IN2_RUNNING	Cannot be cleared because program is executing.
	Other	Emulator error

5.2.81 _cv_clear_cache: Clear coverage cache

Function name:	int _cv_clear_cache()	
Parameter:	None	
Returned value:	TRUE	Return value is always TRUE.
Description:	This function clears the coverage cache.	

5.2.82 _syscom: Execute PD38's script command

Function name:	int _syscom(char *command)		
Parameter:	char	*command	Character string of PD38 script command
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function executes the character string specified by "command" as a script command of PD38. For a script command that dumps a range of addresses from 1000H to 1FFFH, for example, specify this function as follows:

```
_syscom("DumpByte 1000, 1FFF");
```

5.2.83 _doscom: Execute DOS command

Function name:	int _doscom(char *command)		
Parameter:	char	*command	Character string of DOS command
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function executes the character string specified by "command" as a DOS command. For a command that redirects the output result to a TMP file after specifying a /W option for the DIR command of DOS, specify this function as follows:

```
_doscom("DIR /W > TMP");
```

5.2.84 List of Emulator Errors

The table below lists the error numbers that are stored in global variable macro_err when a system call function returns FALSE.

Error number	Description
ER_IN2_MCU_RESET	Target is reset.
ER_IN2_ERROR_2	Checksum error is found in received data
ER_IN2_ERROR_3	Specified data does not exist.
ER_IN2_ERROR_4	Target program is executing.
ER_IN2_ERROR_5	Target program is idle.
ER_IN2_ERROR_6	Measurement has already been stopped.
ER_IN2_ERROR_7	Measurement is already being executed.
ER_IN2_ARG_ERROR	Argument error.
ER_IN2_ERROR_9	Measurement is not completed.
ER_IN2_ERROR_A	No trace data is found for specified cycle.
ER_IN2_ERROR_B	Trace data is nonexistent.
ER_IN2_MCU_DISRESET	Target cannot be reset.
ER_IN2_MCU_POF	MCU power is turned off.
ER_IN2_ERROR_E	Time measurement counter overflowed.
ER_IN2_ERROR_F	POF state was cleared by forcibly resetting.
ER_IN2_ERROR_G	Number of points exceeds valid range.
ER_IN2_ERROR_H	No break is set.
ER_IN2_ERROR_I	No source line information is loaded.
ER_IN2_ERROR_J	Trigger mode is not soft output.
ER_IN2_MCU_CLKOFF	Target clock is turned off.
ER_IN2_ERROR_L	Exception processing was detected during single-stepping.
ER_IN2_ERROR_M	Function range is out of setting.
ER_IN2_ERROR_N	Error in writing to EEPROM.
ER_IN2_MCU_NOSOURCE	Target power is turned off.
ER_IN2_MCU_RUN	Target MCU execution error.
ER_IN2_ERROR_Q	The specified command code is not executable
ER_IN2_ERROR_R	The processor mode and the target system are the disagreements.
ER_IN2_ERROR_S	The specified bank isn't defined in the expansion memory
ER_IN2_ERROR_T	The bank set up is duplicated
ER_IN2_ERROR_U	The specified area includes the debugging monitor memory area
ER_IN2_ERROR_V	The specified area includes the debugging monitor work area

5.3 System Call Functions for Window Operation (winlib.lib)

The winlib.lib provides window-operating functions that can be used in custom window programs. The prototype declaration of each function is written in winlib.h.

Function name	Description
_win_printf	Output text with format included
_win_puts	Output character string to custom window
_win_set_cursor	Set cursor position
_win_set_color	Set text color
_win_set_bkcolor	Set background color
_win_column2dot	Convert cursor coordinates into pixel coordinates
_draw_text_out	Output character string to custom window
_draw_set_color	Set text color
_draw_set_bkcolor	Set background color
_draw_set_bkmode	Set background mode
_draw_set_font	Set font
_draw_get_char_size	Get font size
_draw_line	Draw line
_draw_fill_rect	Fill rectangle
_draw_frame_rect	Draw rectangle
_draw_invert_rect	Reverse rectangle color
_draw_arc	Draw arc of ellipse
_draw_pie	Draw sector
_win_redraw	Redraw custom window
_win_redraw_clear	Redraw custom window
_win_redraw_item	Redraw control item
_win_show_window	Show/hide control item
_win_set_window_title	Set title of custom window
_win_enable_window	Enable/disable control item
_win_button_create	Create button
_win_button_set_text	Change button text
_win_hscroll_range	Set scroll range of horizontal scroll bar
_win_hscroll_pos	Set position of horizontal scroll box
_win_vscroll_range	Set scroll range of vertical scroll bar
_win_vscroll_pos	Set position of vertical scroll box
_win_statusbar_create	Create status bar
_win_statusbar_set_pane	Set items of status bar
_win_statusbar_set_text	Set text of status bar
_win_dialog	Create input dialog box
_win_message_box	Create message box
_win_filedialog	Create file selection dialog box
_win_set_window_pos	Set position of custom window
_win_set_window_size	Set size of custom window
_win_timer_set	Set system timer
_win_timer_kill	Reset system timer

5.3.1 _win_printf: Output text with format included

Function name:	int _win_printf(char *format , ...);		
Parameter:	char	*forma	Format
	...	Variable parameter	
Returned value:	int	Number of characters output	
Description:	This function outputs characters to the cursor position of the custom window after converting them under control of “format” using the text color specified by the _win_set_color() function and the background color specified by the _win_set_bkcolor() function. The cursor is set at a position immediately following the last character that is output. The cursor position can be set at any desired place using the _win_set_cursor() function. Note that only the character font FIXED_SYS can be used.		

5.3.2 _win_puts: Output character string to custom window

Function name:	int _win_puts(char *str)		
Parameter:	char	*str	Output character string
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function outputs a character string specified by str to the cursor position of the customer window using the text color specified by the _win_set_color() function and the background color specified by the _win_set_bkcolor() function. The cursor is set at a position immediately following the last character that is output. The cursor position can be set at any desired place using the _win_set_cursor() function. Note that only the character font FIXED_SYS can be used.		

5.3.3 _win_set_cursor: Set cursor position

Function name:	int _win_set_cursor(int x, int y)		
Parameter:	int	x	Specified x column of cursor
	int	y	Specified y column of cursor
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function moves the cursor to a position specified by x and y. The cursor position is defined with the origin (0, 0) at the upper left corner of the client area of the custom window, the x columns increasing from there to the right and the y columns increasing from there to the bottom. One character is output in one column.		

5.3.4 _win_set_color: Set text color

int _win_set_color(int color)

Parameter:	int	color Text color
Returned value:	int	Previous text color

Description: This function sets a color specified by “color” for text. The text color specified by this function is used when a character string is output using the _win_printf() and the _win_puts() functions. For “color”, specify one of the color constants listed below.

Color constant	Color
COLOR_BLACK	Black
COLOR_BLUE	Blue
COLOR_GREEN	Green
COLOR_CYAN	Sky blue
COLOR_RED	Red
COLOR_MAGENDA	Purple
COLOR_YELLOW	Yellow
COLOR_WHITE	White
COLOR_GRAY	Gray
COLOR_DKBLUE	Dark blue
COLOR_DKGREEN	Dark green
COLOR_DKCYAN	Dark sky blue
COLOR_DKRED	Dark red
COLOR_DKMAGENDA	Dark purple
COLOR_DKYELLOW	Dark yellow
COLOR_LTGRAY	Light gray

5.3.5 _win_set_bkcolor: Set background color

Function name:	int _win_set_bkcolor(int color)
Parameter:	int color Background color of text
Returned value:	int Previous background color
Description:	This function sets a color specified by "color" for the current background. The text color specified by this function is used when a character string is output using the _win_printf() and the _win_puts() functions. For "color", specify one of the color constants listed below.

Color constant	Color
COLOR_BLACK	Black
COLOR_BLUE	Blue
COLOR_GREEN	Green
COLOR_CYAN	Sky blue
COLOR_RED	Red
COLOR_MAGENDA	Purple
COLOR_YELLOW	Yellow
COLOR_WHITE	White
COLOR_GRAY	Gray
COLOR_DKBLUE	Dark blue
COLOR_DKGREEN	Dark green
COLOR_DKCYAN	Dark sky blue
COLOR_DKRED	Dark red
COLOR_DKMAGENDA	Dark purple
COLOR_DKYELLOW	Dark yellow
COLOR_LTGRAY	Light gray

5.3.6 _win_column2dot: Convert cursor coordinates into pixel coordinates

Function name:	int _win_column2dot(int xcol, int ycol, int *xpix, int *ypix)		
Parameter:	int	xcol	X column
	int	ycol	Y column
	int	*xpix	X pixel coordinate of X column position
	int	*ypix	Y pixel coordinate of Y column position
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function converts the cursor coordinates specified by xcol and ycol into pixel coordinates and stores them in *xpix and *ypix.		

5.3.7 _draw_text_out: Output character string to custom window

Function name:	int _draw_text_out(int x, int y, char *str)		
Parameter:	int	x	Logical x coordinate of start point of text
	int	y	Logical y coordinate of start point of text
	char	*str	Pointer to character string to be drawn
Returned value:	TRUE	Return value is always TRUE.	
Description:	Using the currently selected font, this function writes a character string to a specified position using the text color specified by the _draw_set_color() function and the background color specified by the _draw_set_bkcolor() function.		

5.3.8 _draw_set_color: Set text color

Function name:	int _draw_set_color(int color)		
Parameter:	int	color	Text color
Returned value:	int	Previous text color	
Description:	This function sets a color specified by “color” for text. The text color specified by this function is used when a character string is output using the _draw_text_out() function. For “color”, specify one of the color constants listed below.		

Color constant	Color
COLOR_BLACK	Black
COLOR_BLUE	Blue
COLOR_GREEN	Green
COLOR_CYAN	Sky blue
COLOR_RED	Red
COLOR_MAGENDA	Purple
COLOR_YELLOW	Yellow
COLOR_WHITE	White
COLOR_GRAY	Gray
COLOR_DKBLUE	Dark blue
COLOR_DKGREEN	Dark green
COLOR_DKCYAN	Dark sky blue
COLOR_DKRED	Dark red
COLOR_DKMAGENDA	Dark purple
COLOR_DKYELLOW	Dark yellow
COLOR_LTGRAY	Light gray

5.3.9 _draw_set_bkcolor: Set background color

Function name:	int _draw_set_bkcolor(int color)	
Parameter:	int color	Background color of text
Returned value:	int	Previous background color

Description: This function sets a color specified by “color” for the current background. The background color specified by this function is used when a character string is output using the _draw_text_out() function. For “color”, specify one of the color constants listed below.

Color constant	Color
COLOR_BLACK	Black
COLOR_BLUE	Blue
COLOR_GREEN	Green
COLOR_CYAN	Sky blue
COLOR_RED	Red
COLOR_MAGENDA	Purple
COLOR_YELLOW	Yellow
COLOR_WHITE	White
COLOR_GRAY	Gray
COLOR_DKBLUE	Dark blue
COLOR_DKGREEN	Dark green
COLOR_DKCYAN	Dark sky blue
COLOR_DKRED	Dark red
COLOR_DKMAGENDA	Dark purple
COLOR_DKYELLOW	Dark yellow
COLOR_LTGRAY	Light gray

If the background mode is a "Fill" mode, the system fills space between style-specified lines, space between brushed hatch lines, and the background of character cells with the background color.

5.3.10 _draw_set_bkmode: Set background mode

Function name:	int _draw_set_bkmode(int flag)	
Parameter:	int flag	Set mode
Returned value:	TRUE	Return value is always TRUE.

Description: This function sets a background mode. Specify whether you want the area to be filled with the background color before drawing text. If TRUE is specified for flag, the background is filled with the current background color (default). If FALSE is specified for flag, the background is not changed before drawing text.

5.3.11 _draw_set_font: Set font

Function name:	int _draw_set_font(int size, int font)
Parameter:	int size Font size
	int font Font constant
Returned value:	TRUE Return value is always TRUE.
Description:	This function specifies the size and the style of the current drawing font. For “font”, specify some of the following font constants combined with a |.

Font constant	Font style
FONT_FIXED_SYS	"FixedSys"
FONT_MINTYO	" MS mincho"
FONT_GOTHIC	" MS Gothic""
FONT_TIMESNEWROMAN	"Times New Roman"
FONT_CENTURY	"Century"
FONT_ARIAL	"Arial"
FONT_BOLD	Bold
FONT_ITALIC	Italic

5.3.12 _draw_get_char_size: Get font size

Function name:	int _draw_get_char_size(int *pWidth, int *pHeight)
Parameter:	int *pWidth Location where character width is stored
	int *pHeight Location where character height is stored
Returned value:	TRUE Return value is always TRUE.
Description:	This function gets the size of the font character currently being set.

5.3.13 _draw_line: Draw line

Function name:	int _draw_line(int x1, int y1, int x2, int y2, int color)		
Parameter:	int	x1	Starting x coordinate of line
	int	y1	Starting y coordinate of line
	int	x2	Ending x coordinate of line
	int	y2	End y coordinate of line
	int	color	Color of line
Returned value:	TRUE	Return value is always TRUE.	

Description: This function draws a line with a specified color between specified coordinate points. For “color” specify one of the color constants shown below.

Color constant	Color
COLOR_BLACK	Black
COLOR_BLUE	Blue
COLOR_GREEN	Green
COLOR_CYAN	Sky blue
COLOR_RED	Red
COLOR_MAGENDA	Purple
COLOR_YELLOW	Yellow
COLOR_WHITE	White
COLOR_GRAY	Gray
COLOR_DKBLUE	Dark blue
COLOR_DKGREEN	Dark green
COLOR_DKCYAN	Dark sky blue
COLOR_DKRED	Dark red
COLOR_DKMAGENDA	Dark purple
COLOR_DKYELLOW	Dark yellow
COLOR_LTGRAY	Light gray

5.3.14 _draw_fill_rect: Fill rectangle

Function name:	int _draw_fill_rect(int x1, int y1, int x2, int y2, int color)		
Parameter:	int	x1	Upper left x coordinate of rectangle
	int	y1	Upper left y coordinate of rectangle
	int	x2	Lower right x coordinate of rectangle
	int	y2	Lower right y coordinate of rectangle
	int	color	Color with which to fill
Returned value:	TRUE	Return value is always TRUE.	

Description: This function draws a rectangle filled with a specified color with its upper left and lower right corners at specified coordinates. For “color” specify one of the color constants shown below.

Color constant	Color
COLOR_BLACK	Black
COLOR_BLUE	Blue
COLOR_GREEN	Green
COLOR_CYAN	Sky blue
COLOR_RED	Red
COLOR_MAGENDA	Purple
COLOR_YELLOW	Yellow
COLOR_WHITE	White
COLOR_GRAY	Gray
COLOR_DKBLUE	Dark blue
COLOR_DKGREEN	Dark green
COLOR_DKCYAN	Dark sky blue
COLOR_DKRED	Dark red
COLOR_DKMAGENDA	Dark purple
COLOR_DKYELLOW	Dark yellow
COLOR_LTGRAY	Light gray

5.3.15 _draw_frame_rect: Draw rectangle

Function name:	int _draw_frame_rect(int x1, int y1, int x2, int y2, int color)		
Parameter:	int	x1	Upper left x coordinate of rectangle
	int	y1	Upper left y coordinate of rectangle
	int	x2	Lower right x coordinate of rectangle
	int	y2	Lower right y coordinate of rectangle
	int	color	Color of rectangle
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function draws lines to form a rectangle filled with a specified color with its upper left and lower right corners at specified coordinates. For color specify one of the color constants shown below.		

Color constant	Color
COLOR_BLACK	Black
COLOR_BLUE	Blue
COLOR_GREEN	Green
COLOR_CYAN	Sky blue
COLOR_RED	Red
COLOR_MAGENDA	Purple
COLOR_YELLOW	Yellow
COLOR_WHITE	White
COLOR_GRAY	Gray
COLOR_DKBLUE	Dark blue
COLOR_DKGREEN	Dark green
COLOR_DKCYAN	Dark sky blue
COLOR_DKRED	Dark red
COLOR_DKMAGENDA	Dark purple
COLOR_DKYELLOW	Dark yellow
COLOR_LTGRAY	Light gray

5.3.16 _draw_invert_rect: Reverse rectangle color

Function name:	int _draw_invert_rect(int x1, int y1, int x2, int y2)		
Parameter:	int	x1	Upper left x coordinate of rectangle
	int	y1	Upper left y coordinate of rectangle
	int	x2	Lower right x coordinate of rectangle
	int	y2	Lower right y coordinate of rectangle
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function reverses the color of the rectangle with its upper left and lower right corners at specified coordinates.		

5.3.17 _draw_arc: Draw arc of ellipse

Function name:	int _draw_arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4, int color)		
Parameter:	int	x1	Upper left x coordinate of boundary rectangle (logical unit)
	int	y1	Upper left y coordinate of boundary rectangle (logical unit)
	int	x2	Lower right x coordinate of boundary rectangle (logical unit)
	int	y2	Lower right y coordinate of boundary rectangle (logical unit)
	int	x3	x coordinate of starting point to draw arc (logical unit)
	int	y3	y coordinate of starting point to draw arc (logical unit)
	int	x4	x coordinate of ending point to draw arc (logical unit)
	int	y4	y coordinate of ending point to draw arc (logical unit)
	int	color	Color of arc
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function draws an arc of a ellipse. Specify the coordinates of a boundary rectangle (x1, y1) and (x2, y2) and the starting point (x3, y3) and ending point (x4, y4) of an arc. The starting and ending points of an arc do not need to be on a line of arc. A line that links a specified starting point and the center of a boundary rectangle is calculated and the starting point of an arc is calculated from it. The ending point is calculated in the same way. For “color” specify one of the color constants shown below.

Color constant	Color
COLOR_BLACK	Black
COLOR_BLUE	Blue
COLOR_GREEN	Green
COLOR_CYAN	Sky blue
COLOR_RED	Red
COLOR_MAGENDA	Purple
COLOR_YELLOW	Yellow
COLOR_WHITE	White
COLOR_GRAY	Gray
COLOR_DKBLUE	Dark blue
COLOR_DKGREEN	Dark green
COLOR_DKCYAN	Dark sky blue
COLOR_DKRED	Dark red
COLOR_DKMAGENDA	Dark purple
COLOR_DKYELLOW	Dark yellow
COLOR_LTGRAY	Light gray

5.3.18 _draw_pie: Draw sector

Function name:	int _draw_pie(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4, int framecolor, int paintcolor)		
Parameter:	int	x1	Upper left x coordinate of boundary rectangle (logical unit)
	int	y1	Upper left y coordinate of boundary rectangle (logical unit)
	int	x2	Lower right x coordinate of boundary rectangle (logical unit)
	int	y2	Lower right y coordinate of boundary rectangle (logical unit)
	int	x3	x coordinate of starting point to draw sector (logical unit)
	int	y3	y coordinate of starting point to draw sector (logical unit)
	int	x4	x coordinate of ending point to draw sector (logical unit)
	int	y4	y coordinate of ending point to draw sector (logical unit)
	int	framecolor	Color of framing line of sector
	int	paintcolor	Color with which to fill sector
Returned value:	TRUE	Succeeded	
	FALSE	Error	

Description: This function draws a sector. Define the circumferential circle of a sector by the boundary rectangle of an ellipse (x1, y1) and (x2, y2). For framecolor and paintcolor, specify the following color constants.

Color constant	Color
COLOR_BLACK	Black
COLOR_BLUE	Blue
COLOR_GREEN	Green
COLOR_CYAN	Sky blue
COLOR_RED	Red
COLOR_MAGENDA	Purple
COLOR_YELLOW	Yellow
COLOR_WHITE	White
COLOR_GRAY	Gray
COLOR_DKBLUE	Dark blue
COLOR_DKGREEN	Dark green
COLOR_DKCYAN	Dark sky blue
COLOR_DKRED	Dark red
COLOR_DKMAGENDA	Dark purple
COLOR_DKYELLOW	Dark yellow
COLOR_LTGRAY	Light gray

5.3.19 _win_redraw: Redraw custom window

Function name:	int _win_redraw()
Parameter:	None
Returned value:	TRUE Return value is always TRUE.
Description:	This function redraws a custom window without erasing its display.

5.3.20 _win_redraw_clear: Redraw custom window

Function name:	int _win_redraw_clear()
Parameter:	None
Returned value:	TRUE Return value is always TRUE.
Description:	This function redraws a custom window after erasing its display.

5.3.21 _win_redraw_item: Redraw control item

Function name:	int _win_redraw_item(int handle)
Parameter:	int handle Handle of control item
Returned value:	TRUE Return value is always TRUE.
Description:	This function redraws a control item specified by “handle” (e.g., button).

5.3.22 _win_show_window: Show/hide control item

Function name:	int _win_show_window(int handle, int flag)
Parameter:	int handle Handle of control item
	int flag TRUE: Displayed FALSE: Not displayed
Returned value:	TRUE Return value is always TRUE.
Description:	This function specifies whether or not to display a control item specified by “handle” (e.g., button). The specified control item is displayed when TRUE is specified for “flag” and is not displayed when FALSE is specified.

5.3.23 _win_set_window_title: Set title of custom window

Function name:	int _win_set_window_title(char *title)
Parameter:	char *title Window title
Returned value:	TRUE Return value is always TRUE.
Description:	This function sets a character string specified by “title” in the title of a custom window.

5.3.24 _win_enable_window: Enable/disable control item

Function name:	int _win_enable_window(int handle, int flag)
Parameter:	int handle Handle of control item
	int flag TRUE: Enabled FALSE: Disabled
Returned value:	TRUE Return value is always TRUE.
Description:	This function specifies a state of the control item specified by “handle” (e.g., button). The specified control item is enabled when TRUE is specified for “flag” and is disabled when FALSE is specified. When disabled, the control item is displayed in gray.

5.3.25 _win_button_create: Create button

Function name:	int _win_button_create(int x1,int y1,int x2,int y2, char *str,int id)		
Parameter:	int	x1	Upper left x coordinate of button
	int	y1	Upper left y coordinate of button
	int	x2	Lower right x coordinate of button
	int	y2	Lower right y coordinate of button
	char	*str	Button control text
	int	id	Button control ID
Returned value:	int	Handle of button	
Description:	This function creates a button in an area specified by x1, y1, x2, and y2 that displays the text specified by str on its surface. The control ID specified by “id” is sent to message handler as the argument nID of the OnCommand() handle function when the button is clicked.		

5.3.26 _win_button_set_text: Change button text

Function name:	int _win_button_set_text(int handle. char *text)		
Parameter:	int	handle	Handle of button
	char	*text	Button control text
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function changes the text displayed on the button specified by “handle” to one that is specified by text.		

5.3.27 _win_hscroll_range: Set scroll range of horizontal scroll bar

Function name:	int _win_hscroll_range(int min, int max)		
Parameter:	int	min	Minimum scroll position of horizontal scroll bar
	int	max	Maximum scroll position of horizontal scroll bar
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function specifies the minimum and maximum scroll positions of the horizontal scroll bar of a custom window. If 0 is specified for both min and max, the horizontal scroll bar is not displayed. By default, the horizontal scroll bar is hidden, with both parameters set to 0. The recommended scroll range is 0 to 100.		

5.3.28 _win_hscroll_pos: Set position of horizontal scroll box

Function name:	int _win_hscroll_pos(int pos)		
Parameter:	int	pos	New position of horizontal scroll box
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function sets the current position of the horizontal scroll box of a custom window and redraws the scroll bar to make it fit the new position of the horizontal scroll box. The new position must be within the scroll range.		

5.3.29 _win_vscroll_range: Set scroll range of vertical scroll bar

Function name:	int _win_vscroll_range(int min, int max)		
Parameter:	int	min	Minimum scroll position of vertical scroll bar
	int	max	Maximum scroll position of vertical scroll bar
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function specifies the minimum and maximum scroll positions of the vertical scroll bar of a custom window. If 0 is specified for both “min” and max, the vertical scroll bar is not displayed. By default, the vertical scroll bar is hidden, with both parameters set to 0. The recommended scroll range is 0 to 100.		

5.3.30 _win_vscroll_pos: Set position of vertical scroll box

Function name:	int _win_vscroll_pos(int pos)		
Parameter:	int	pos	New position of vertical scroll box
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function sets the current position of the vertical scroll box of a custom window and redraws the scroll bar to make it fit the new position of the vertical scroll box. The new position must be within the scroll range.		

5.3.31 _win_statusbar_create: Create status bar

Function name:	int _win_statusbar_create(int cnt)		
Parameter:	int	cnt	Number of items on status bar
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function creates a status bar at bottom of a custom window. For cnt, set the number of items on this status bar.		

5.3.32 _win_statusbar_set_pane: Set items of status bar

Function name:	int _win_statusbar_set_pane(int index, int style, int size)		
Parameter:	int	index	Index number of status bar item
	int	style	Style of item
	int	size	Size of item (in pixels)
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function sets the style specified by "style" and the size specified by "size" for the item on the created status bar that is specified by "index". For style, specify one of the styles shown below.		

Style	Description
SBPS_NOBORDERS	Does not have 3D boundary line round pane.
SBPS_POPOUT	Has boundary line displayed in inverse video with text raised to the surface.
SBPS_DISABLED	Does not draw text.
SBPS_NORMAL	Neither stretched nor inverted. Does not have boundary line either.
SBPS_STRETCH	Stretches pane to fill unused space. Only one pane of this style is allowed for the status bar. This style can be combined with some other style using a |.

5.3.33 _win_statusbar_set_text: Set text of status bar

Function name:	int _win_statusbar_set_text(tint index, char *text)		
Parameter:	int	index	Index number of status bar item
	char	*text	Text displayed on status bar
Returned value:	TRUE	Return value is always TRUE.	
Description:	This function sets text to be displayed in a status bar item.		

5.3.34 _win_dialog: Create input dialog box

Function name:	int _win_dialog(char *str, char *buf)		
Parameter:	char	*str	Character string for message to be displayed
	char	*buf	Location where obtained character string is stored
Returned value:	TRUE	OK button is pressed	
	FALSE	Cancel button is pressed	
Description:	This function opens an input dialog box and gets one line of character string.		

5.3.35 _win_message_box: Create message box

Function name:	int _win_message_box(char *str, char *title, int style)		
Parameter:	char	*str	Message to be displayed
	char	*title	Title of message box
	int	style	Operation and content of message box
Returned value:	int	Execution result of functions shown below	

Value	Meaning
0	No sufficient memory
IDABORT	[Stop] button selected
IDCANCEL	[Cancel] button selected
IDIGNORE	[Ignore] button selected
IDNO	[No] button selected
IDOK	[OK] button selected
IDRETRY	[Retry] button selected
IDYES	[Yes] button selected

Description: This function creates a message box. For style, specify the following styles combined with a |.

Style	Description
MB_ABORTRETRYIGNORE	Message box contains three pushbuttons: [Stop], [Retry], and [Ignore].
MB_APPLMODAL	Operation of PD38/CB38 is stopped until message box is responded (default).
MB_DEFBUTTON1	First button is the default. The first button is always the default unless MB_DEFBUTTON2 or MB_DEBUTTON3 is specified.
MB_DEFBUTTON2	Second button is the default.
MB_DEFBUTTON3	Third button is the default.
MB_ICONEXCLAMATION	Exclamation mark icon is displayed in the message box.
MB_ICONHAND	Same as MB_ICONSTOP.
MB_ICONINFORMATION	Icon with lowercase "i" in a circle is displayed in the message box.
MB_ICONQUESTION	Question mark (?) icon is displayed in the message box.
MB_ICONSTOP	[STOP] icon is displayed in the message box.
MB_OK	Message box contains an [OK] pushbutton.
MB_OKCANCEL	Message box contains [OK] and [Cancel] pushbuttons.
MB_RETRYCANCEL	Message box contains [Retry] and [Cancel] pushbuttons
MB_SYSTEMMODAL	All applications are suspended until the user responds to the message box. Use this message box to inform serious and potentially dangerous errors (e.g., memory shortage) that require immediate corrective action.

Style(continued from preceding page)	Description
MB_YESNO	Message box contains two pushbuttons: [Yes] and [No].
MB_YESNOCANCEL	Message box contains three pushbuttons: [Yes], [No], and [Cancel].

5.3.36 _win_filedialog: Create file selection dialog box

Function name int _win_filedialog(char *title, int openFileDialog,
char *defExt, char *defFileName, int flags,
char *filter, char *fileName)

Parameter:	char	*title	Title of dialog box
	int	openFileDialog	Specification to open or save
	char	*defExt	Default file name extension
	char	*defFileName	Default file name
	int	flags	Flag to customize dialog box
	char	*filter	Specify a filter
	char	*fileName	Destination where acquired file name is store

Returned value:	TRUE	OK button was pressed.
	FALSE	Cancel button was pressed.

Description: This function creates a file selection dialog box and gets a selected file name. For “title”, specify the title of the dialog box. For openFileDialog, specify TRUE when building a dialog box to "Open a file" and FALSE when building a dialog box to "Save file after giving it a name." For “defExt”, specify a file name extension you want to be automatically added when a file name is input in the file name edit box without adding an extension. No extension is added if you specify NULL here. For defFileName, specify the file name that is displayed first in the file name entering edit box. No file name is displayed if you specify NULL here. For “flags”, specify the styles shown below by combining them with |.

Flag	Description
OFN_ALLOWMULTISELECT	This flag specifies that multiple choices can be selected in the "File name" list box. (When you create a dialog box using a private template, the LBS_EXTENDEDSEL value must be specified in the definition of the "File name" list box.)

Flag	Description
OFN_CREATEPROMPT	This flag specifies that if a specified file cannot be found, the user be asked to confirm whether a new file need be created by the dialog box function. (This flag sets the OFN_PATHMUSTEXIST and OFN_FILEMUSTEXIST flags automatically.)
OFN_FILEMUSTEXIST	This flag specifies that the user can only input an existing file name in the "File name" entry field. If an invalid file name is input in the "File name" entry field by the user when this flag is set, the dialog box function displays a warning in the message box. When this flag is set, the OFN_PATHMUSTEXIST is set also.
OFN_HIDEREADONLY	This flag turns off (hides) the [Read-only] check box.
OFN_NOCHANGEDIR	This flag directs the dialog box to reset the current directory to one that was selected when calling the dialog box.
OFN_NONETWORKBUTTON	This flag turns off the [Network] button to disable it from being used.
OFN_NOREADONLYRETURN	This flag specifies that the [Read-only] check box of the returned file be not checked, and that the file be not placed in a write-protected directory.
OFN_NOTESTFILECREATE	This flag specifies that a file be not created before closing the dialog box. This flag must be set if the application saves a file in the network-shared point that is "Created but not corrected." If the application sets this flag, the library does no longer check whether the file is write-protected, disk capacity is available, the drive door is open, and whether the network is protected. Once the file is closed while in this state, it cannot be reopened. Therefore, applications that use this flag must handle files with caution.
OFN_OVERWRITEPROMPT	If a selected file already exists, this flag causes the dialog box for "Saving file after giving it a name" to generate a message box. The user must confirm whether the file can be overwritten.

Flag	Description
OFN_PATHMUSTEXIST	This flag specifies that the user can only input a valid path. If an invalid path is input in the "File name" entry field by the user when this flag is set, the dialog box function displays a warning in the message box.
OFN_READONLY	When creating a dialog box, this flag ensures that the [Read-only] check box by default is checked. It also indicates the status of the [Read-only] check box when the dialog box is closed.

For filter, specify a pair of character strings to specify the filters that identify a file by using the format shown below. In the example below, filters (*.m;*.h) and (*.*) are specified.

"Files(*.m;*.h)|*.m;*.h|All Files(*.*)|*.*||"

Once filters are specified, the file list box displays only the selected ones, with others gone. The selected file name is stored in FileName. If multiple files are selected in cases when selection of multiple files is allowed, the space character is stored as the delimiter.

5.3.37 _win_set_window_pos: Set position of custom window

Function name:	int _win_set_window_pos(int x, int y)		
Parameter:	int	x	New left-side position of custom window
	int	y	New upper-side position of custom window
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function changes the position of a custom window.		

5.3.38 _win_set_window_size: Set size of custom window

Function name:	int _win_set_window_size(int cx, int cy)		
Parameter:	int	cx	New width of custom window
	int	cy	New height of custom window
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function changes the size of a custom window.		

5.3.39 _win_timer_set: Set system timer

Function name:	int _win_timer_set(int nId, int nElapse)		
Parameter:	int	nId	Timer identifier other than 0
	int	nElapse	Time-out value (in ms)
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function sets a system timer that has the timer identifier specified by nID. A time-out value is specified, sot that each time the timer times out, the system stores the timer identifier value in parameter nIDEvent and calls the OnTimer() handler function. To reset the timer, use the _win_timer_kill() function.		

5.3.40 _win_timer_kill: Reset system timer

Function name:	int _win_timer_kill(int nId)		
Parameter:	int	nId	Timer identifier other than 0
Returned value:	TRUE	Succeeded	
	FALSE	Error	
Description:	This function resets the system timer specified by nID.		

5.4 Handle Functions for Custom Window

Handle functions are written in a framework that is automatically generated by CB38 when creating a new project in the custom window creation mode. These functions are called when a custom window receives a message from Windows. The table below lists the handle functions.

Handle function name	Description
OnChar	When a key that can be converted into ASCII character code is pressed, this function is called following the OnKeyDown() handle function.
OnCommand	Called when command message is received.
OnCreate	Called when window creation is requested.
OnDestroy	Called when window destruction is requested.
OnDraw	Called when window redrawing is requested.
OnEvent	Called when PD38 event is received.
OnHScroll	Called when horizontal scroll bar is clicked.
OnKeyDown	Called when a key other than system keys is pressed.
OnKeyUp	Called when a key other than system keys is released.
OnLButtonDblClk	Called when left mouse button is double-clicked.
OnLButtonDown	Called when left mouse button is pressed.
OnLButtonUp	Called when left mouse button is released.
OnMouseMove	Called when mouse cursor is moved.
OnRButtonDblClk	Called when right mouse button is double-clicked.
OnRButtonDown	Called when right mouse button is pressed.
OnRButtonUp	Called when right mouse button is released.
OnSize	Called when window size is changed.
OnTimer	Called when time-out interval is informed due to elapsed time of timer.
OnVScroll	Called when vertical scroll bar is clicked.

5.4.1 Specifications of Data Passed to Handle Functions

A handle function is called when the custom window receives a message from Windows. When calling a handle function, the custom window stores the information attached to the message in an area indicated by global variable _HandleMsgBlock to make it referencible from the handle function.

The following shows an example of how information is passed to a handle function via global variable _HandleMsgBlock.

```
extern  char    _HandleMsgBlock[32];

OnSize()
{
        int     nType;  /* Message data */
        int     cx:;    /* Message data */
        int     cy;     /* Message data */

        /* Restore message data */
        nType = ((int*)_HandleMsgBlock)[0];
        cx = ((int*)_HandleMsgBlock)[1];
        cy = ((int*)_HandleMsgBlock)[2];

        /* Write message handler code hear, please. */

}
```

At the beginning of a handle function, the information stored in _HandleMsgBlock is stored in a local variable of the handle function. Once this processing is made, the information passed to the handle function can be referenced as a variable.
The information passed to handle functions varies with each handle function. The contents of these processing are written in framework by default.

5.4.2 OnChar Handle Function

Function name: OnChar

Description: When a key that can be converted into ASCII character code is pressed, this function is called following the OnKeyDown() handle function.

Data: The information stored in _HandleMsgBlock is shown below:

ASCII character code	4 bytes
Repeat count	4 bytes
Flag(unused)	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int	nChar	ASCII character code value
int	nRepCnt	Repeat count value indicating a number of times a key stroke is generated while the key is held down.
int	nFlags	Not used in this version.

5.4.3 OnCommand Handle Function

Function name: OnCommand

Description: This function is called when a command message is received from Windows.

Data: The information stored in _HandleMsgBlock is shown below:

Command ID	4 bytes
Advice message	4 bytes
Handle	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int	nId	Command ID of control item
int	nMsg	Advice message of control item
int	nHandle	Handle of control item

Supplement: This handle function is called mainly when an event occurs in the control items set for the custom window. The ID number to identify the control item is set in nID; the advice message to identify the encountered event is set in nMsg; and the handle of the control item is set in nHandle. The values set in these variables differ with each control item. For details, refer to specifications of the system call functions that are used to manipulate the control items.

5.4.4 OnCreate Handle Function

Function name: OnCreate

Description: This function is called when a request to create a window is received. This function performs such operations as to generate control items, etc. and to initialize variables.

Data: None

Variables: None

5.4.5 OnDestroy Handle Function

Function name: OnDestroy

Description: This function is called when a request to destroy a window is received. This function performs such operations as to free an allocated heap area.

Data: None

Variables: None

5.4.6 OnDraw Handle Function

Function name: OnDraw

Description: This function is called when a request to redraw a window is received. The cases where this function is called are when it is necessary to display part of a window that is hidden by some other window. This function performs such operations as to redraw a custom window.

Data: None

Variables: None

5.4.7 OnEvent Handle Function

Function name:	OnEvent
Description:	This function is called when a PD38 event is received from PD38. The cases where this function is called are when it is necessary to change the PD38 status. This function performs such operations as to get memory values and redraw a window as necessary.
Data:	The information stored in _HandleMsgBlock is shown below:

PD38 event number	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nEventID PD38 event numbers listed below

PD38 event number	Cases when event is received
EVENT_GO	Start of execution
EVENT_STOP	Stop of execution
EVENT_RESET	Reset
EVENT_STEP	Execution of Step command
EVENT_OVER	Execution of Over command
EVENT_RETURN	Execution of Return command
EVENT_PUT_REG	Change of register value
EVENT_REG_PC	Change of PC value
EVENT_PUT_MEM	Change of memory value
EVENT_LOAD	Program load
EVENT_ADD_SYMBOL	Addition of assembler symbol
EVENT_DEL_SYMBOL	Deletion of assembler symbol
EVENT_SBRK	Change of software breakpoint
EVENT_TRACE_START	Start of trace measurement
EVENT_TRACE_END	End of trace measurement
EVENT_TRACE_PASS	Passage of trace point
EVENT_FUNC	Change of displayed function
EVENT_FILE	Change of displayed file
EVENT_UP	Change of scope to high-level function
EVENT_DOWN	Change of scope to low-level function
EVENT_MAP	Change of map
EVENT_PATH	Change of search path
EVENT_RAMDISP	Redrawing of real-time RAM monitor
EVENT_RAMINFO	Redrawing of real-time RAM monitor
EVENT_HWBRK	Change of hardware break settings
EVENT_EXIT	Termination of PD38
EVENT_FONT	Change of font
EVENT_TAB	Change of tabstop value
EVENT_CWATCH_UPDATE	Redrawing of C watch window
EVENT_SCRIPT_INIT	Initialization of script window
EVENT_TIME_10MS	Timer interrupt at 10 ms intervals

5.4.8 OnHScroll Handle Function

Function name:	OnHScroll
Description:	This function is called when the horizontal scroll bar is clicked.
Data:	The information stored in _HandleMsgBlock is shown below:

Scroll bar code	4 bytes
Position of scroll box	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int	nSBCode	Scroll bar code indicating one of the following scroll requests

Value	Description
SB_LEFT	Scroll to left edge
SB_ENDSCROLL	End of scroll
SB_LINELEFT	Scroll to left
SB_LINERIGHT	Scroll to right
SB_PAGELEFT	Scroll one page to left
SB_PAGERIGHT	Scroll one page to right
SB_RIGHT	Scroll to right edge
SB_THUMBPOSITION	Scroll to absolute position (current position specified by nPos)
SB_THUMBTRACK	Drag scroll box to specified position (current position specified by “nPos”)

int	nPos	Position when “nSBCode” is SB_THUMBPOSITION or SB_THUMBTRACK.

5.4.9 OnKeyDown Handle Function

Function name: OnKeyDown

Description: This function is called when a key is pressed. However, the keys that belong to the "system keys" do not have any effect. Although the "system keys" are defined differently depending on the type of personal computer, they normally consist of the Alt key and some other key that is entered simultaneously with the Alt key.

Data: The information stored in _HandleMsgBlock is shown below:

Virtual key code	4 bytes
Repeat count	4 bytes
Flag	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int	nChar	Virtual key code value of key
int	nRepCnt	Repeat count value indicating a number of times a key stroke is generated while the key is held down.
int	nFlags	One of the following status flags

Bit	Description
0 to 7	Unused.
8	Extension key. Function keys and keys on numeric keypad. (This bit is 1 for extended keys; otherwise, 0.)
11 to 12	Unused.
13	Always 0.
14	Immediately preceding key status. (This bit is 1 when a key is pressed when called; otherwise, 0.)
15	Always 0.

For details about virtual key code, refer to "About virtual key code" in the next page.

[About virtual key code]
To support all models available, Windows has virtual keys defined to the actual keys on the keyboard. For example, when depression of the F1 key is detected, Windows converts it into the virtual key code that corresponds to the F1 key and informs depression of the F1 key to the application. Thanks to the use of virtual keys, the application need not be concerned with the difference in the keyboard.
In CB38, the following virtual key codes can be used.

Virtual key code	Corresponding key on keyboard
VK_CANCEL	Ctrl + Break
VK_BACK	Backspace
VK_TAB	Tab
VK_CLEAR	5 on numeric keypad when Num Lock is off
VK_RETURN	Enter
VK_SHIFT	Shift
VK_CONTROL	Ctrl
VK_MENU	Alt
VK_PAUSE	Pause
VK_CAPITAL	Casp Lock
VK_ESCAPE	Esc
VK_SPACE	Spasebar
VK_PRIOR	Page Up
VK_NEXT	Page Down
VK_END	End
VK_HOME	Home
VK_LEFT	<-
VK_UP	Up
VK_RIGHT	->
VK_DOWN	Down
VK_SNAPSHOT	Print Screen
VK_INSERT	Ins
VK_DELETE	Del
VK_NUMPAD0	0 on numeric keypad when Num Lock is on
VK_NUMPAD1	1 on numeric keypad when Num Lock is on
VK_NUMPAD2	2 on numeric keypad when Num Lock is on
VK_NUMPAD3	3 on numeric keypad when Num Lock is on
VK_NUMPAD4	4 on numeric keypad when Num Lock is on
VK_NUMPAD5	5 on numeric keypad when Num Lock is on
VK_NUMPAD6	6 on numeric keypad when Num Lock is on
VK_NUMPAD7	7 on numeric keypad when Num Lock is on
VK_NUMPAD8	8 on numeric keypad when Num Lock is on
VK_NUMPAD9	9 on numeric keypad when Num Lock is on

Virtual key code	Corresponding key on keyboard
VK_MULTIPLY	$\boxed{*}$ on numeric keypad (extended keyboard)
VK_ADD	$\boxed{+}$ on numeric keypad (extended keyboard)
VK_SUBTRACT	$\boxed{-}$ on numeric keypad (extended keyboard)
VK_DIVIDE	$\boxed{/}$ on numeric keypad (extended keyboard)
VK_F1	Function key $\boxed{F1}$
VK_F2	Function key $\boxed{F2}$
VK_F3	Function key $\boxed{F3}$
VK_F4	Function key $\boxed{F4}$
VK_F5	Function key $\boxed{F5}$
VK_F6	Function key $\boxed{F6}$
VK_F7	Function key $\boxed{F7}$
VK_F8	Function key $\boxed{F8}$
VK_F9	Function key $\boxed{F9}$
VK_F10	Function key $\boxed{F10}$
VK_F11	Function key $\boxed{F11}$ (extended keyboard)
VK_F12	Function key $\boxed{F12}$ (extended keyboard)¥
VK_NUMLOCK	$\boxed{\text{Num Lock}}$
VK_SCROLL	$\boxed{\text{Scroll Lock}}$

For keys $\boxed{0}$ to $\boxed{9}$ and keys $\boxed{A}$ to $\boxed{Z}$, virtual key code values "0" to "9" and values "A" to "Z" are used, respectively.

5.4.10 OnKeyUp Handle Function

Function name: OnKeyUp

Description: This function is called when a key is released. However, the keys that belong to the "system keys" do not have any effect. Although the "system keys" are defined differently depending on the type of personal computer, they normally consist of the Alt key and some other key that is entered simultaneously with the Alt key.

Data: The information stored in _HandleMsgBlock is shown below:

Virtual key code	4 bytes
Repeat count	4 bytes
Flag	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int	nChar	Virtual key code value of key
int	nRepCnt	Repeat count value that indicates the number of times the key stroke is generated while the key is held down. This value is 1 when the OnKeyUp handle function is called.
int	nFlags	One of the following status flags

Bit	Description
0-7	Unused.
8	Extension key. Function keys and keys on numeric keypad. (This bit is 1 for extended keys; otherwise, 0.)
11 to 12	Unused.
13	Always 0.
14	Immediately preceding key status. (This bit is 1 when a key is pressed when called; otherwise, 0.)
15	Always 0.

For details about virtual key code, refer to "About virtual key code" in the preceding page.

5.4.11 OnLButtonDblClk Handle Function

Function name: OnLButtonDblClk

Description: This function is called when the left mouse button is double-clicked.

Data: The information stored in _HandleMsgBlock is shown below:

Type of virtual key	4 bytes
x coordinate of cursor	4 bytes
y coordinate of cursor	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nFlags Virtual key that is pressed
The stored value is a logical sum of the following values representing a virtual key.

Value	Description
MK_CONTROL	Ctrl key pressed
MK_LBUTTON	Left mouse button pressed
MK_MBUTTON	Middle mouse button pressed
MK_RBUTTON	Right mouse button pressed
MK_SHIFT	Shift key pressed

int x x coordinate of mouse cursor

int y y coordinate of mouse cursor

Coordinates are always a relative position referenced to the upper left corner of the window.

5.4.12 OnLButtonDown Handle Function

Function name: OnLButtonDown

Description: This function is called when the left mouse button is pressed.

Data: The information stored in _HandleMsgBlock is shown below:

Type of virtual key	4 bytes
x coordinate of cursor	4 bytes
y coordinate of cursor	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nFlags Virtual key that is pressed
The stored value is a logical sum of the following values representing a virtual key.

Value	Description
MK_CONTROL	Ctrl key pressed
MK_LBUTTON	Left mouse button pressed
MK_MBUTTON	Middle mouse button pressed
MK_RBUTTON	Right mouse button pressed
MK_SHIFT	Shift key pressed

int x x coordinate of mouse cursor

int y y coordinate of mouse cursor

Coordinates are always a relative position referenced to the upper left corner of the window.

5.4.13 OnLButtonUp Handle Function

Function name: OnLButtonUp

Description: This function is called when the left mouse button is released.

Data: The information stored in _HandleMsgBlock is shown below:

Type of virtual key	4 bytes
x coordinate of cursor	4 bytes
y coordinate of cursor	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nFlags Virtual key that is pressed
The stored value is a logical sum of the following values representing a virtual key.

Value	Description
MK_CONTROL	Ctrl key pressed
MK_LBUTTON	Left mouse button pressed
MK_MBUTTON	Middle mouse button pressed
MK_RBUTTON	Right mouse button pressed
MK_SHIFT	Shift key pressed

int x x coordinate of mouse cursor

int y y coordinate of mouse cursor

Coordinates are always a relative position referenced to the upper left corner of the window.

5.4.14 OnMouseMove Handle Function

Function name: OnMouseMove

Description: This function is called when the mouse cursor is moved.

Data: The information stored in _HandleMsgBlock is shown below:

Type of virtual key	4 bytes
x coordinate of cursor	4 bytes
y coordinate of cursor	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nFlags Virtual key that is pressed
The stored value is a logical sum of the following values representing a virtual key.

Value	Description
MK_CONTROL	Ctrl key pressed
MK_LBUTTON	Left mouse button pressed
MK_MBUTTON	Middle mouse button pressed
MK_RBUTTON	Right mouse button pressed
MK_SHIFT	Shift key pressed

int x x coordinate of mouse cursor

int y y coordinate of mouse cursor

Coordinates are always a relative position referenced to the upper left corner of the window.

5.4.15 OnRButtonDblClk Handle Function

Function name: OnRButtonDblClk

Description: This function is called when the right mouse button is double-clicked

Data: The information stored in _HandleMsgBlock is shown below:

Type of virtual key	4 bytes
x coordinate of cursor	4 bytes
y coordinate of cursor	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nFlags Virtual key that is pressed
The stored value is a logical sum of the following values representing a virtual key.

Value	Description
MK_CONTROL	Ctrl key pressed
MK_LBUTTON	Left mouse button pressed
MK_MBUTTON	Middle mouse button pressed
MK_RBUTTON	Right mouse button pressed
MK_SHIFT	Shift key pressed

int x x coordinate of mouse cursor
int y y coordinate of mouse cursor
Coordinates are always a relative position referenced to the upper left corner of the window.

5.4.16 OnRButtonDown Handle Function

Function name: OnRButtonDown

Description: This function is called when the right mouse button is pressed.

Data: The information stored in _HandleMsgBlock is shown below:

Type of virtual key	4 bytes
x coordinate of cursor	4 bytes
y coordinate of cursor	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nFlags Virtual key that is pressed
The stored value is a logical sum of the following values representing a virtual key.

Value	Description
MK_CONTROL	Ctrl key pressed
MK_LBUTTON	Left mouse button pressed
MK_MBUTTON	Middle mouse button pressed
MK_RBUTTON	Right mouse button pressed
MK_SHIFT	Shift key pressed

int x x coordinate of mouse cursor
int y y coordinate of mouse cursor
Coordinates are always a relative position referenced to the upper left corner of the window.

5.4.17 OnRButtonUp Handle Function

Function name: OnRButtonUp

Description: This function is called when the right mouse button is released.

Data: The information stored in _HandleMsgBlock is shown below:

Type of virtual key	4 bytes
x coordinate of cursor	4 bytes
y coordinate of cursor	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nFlags Virtual key that is pressed
The stored value is a logical sum of the following values representing a virtual key.

Value	Description
MK_CONTROL	Ctrl key pressed
MK_LBUTTON	Left mouse button pressed
MK_MBUTTON	Middle mouse button pressed
MK_RBUTTON	Right mouse button pressed
MK_SHIFT	Shift key pressed

int x x coordinate of mouse cursor

int y y coordinate of mouse cursor

Coordinates are always a relative position referenced to the upper left corner of the window.

5.4.18 OnSize Handle Function

Function name: OnSize

Description: This function is called when the window size is changed.

Data: The information stored in _HandleMsgBlock is shown below:

Type of size change	4 bytes
New width	4 bytes
New height	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nType One of the following types of size changes that is requested

Value	Description
SIZE_MAXIMIZED	Maximized display
SIZE_MINIMIZED	Iconification
SIZE_RESTORED	Size changed, but SIZE_MINIMIZED and SIZE_MAXIMIZED are not applied.
SIZE_MAXHIDE	Message is sent to all pup-up windows when several other windows are maximized in size.
SIZE_MAXSHOW	Message is sent to all pup-up windows when several other windows are restored to previous size.

int cx New width of client area

int cy New height of client area

5.4.19 OnTimer Handle Function

Function name: OnTimer

Description: This function is called when a time-out interval is informed due to an elapsed time of the timer.

Data: The information stored in _HandleMsgBlock is shown below:

Timer identifier	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int nIDEvent Identification number of timer

5.4.20 OnVScroll Handle Function

Function name:	OnVScroll
Description:	This function is called when the vertical scroll bar is clicked.
Data:	The information stored in _HandleMsgBlock is shown below:

Scroll bar code	4 bytes
Position of scroll box	4 bytes

Variables: The variables set by _HandleMsgBlock are shown below.

int	nSBCode	Scroll bar code indicating one of the following scroll requests

Value	Description
SB_BOTTOM	Scroll to bottom
SB_ENDSCROLL	End of scroll
SB_LINEDOWN	Scroll one line down
SB_LINEUP	Scroll one line up
SB_PAGEDOWN	Scroll one page down
SB_PAGEUP	Scroll one page up
SB_THUMBPOSITION	Scroll to absolute position (current position specified by nPos)
SB_THUMBTRACK	Drag scroll box to specified position (current position specified by nPos)
SB_TOP	Scroll to top

int	nPos	Position when “nSBCode” is SB_THUMBPOSITION or SB_THUMBTRACK.

[MEMO]

CB38 V.1.00 User's Manual

Rev. 1.00
May 1, 2003
REJ10J0093-0100Z

CB38 V.1.00
User's Manual

Renesas Electronics Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki-shi, Kanagawa 211-8668 Japan

REJ10J0093-0100Z