Pocket Viewer (PV-S1600)
Software Development Kit
User’s MANUAL

Ver 1.00

(Oct. 1st 2002)

CASIO COMPUTER CO.,LTD.

Contents

1. Before You Begin

1) This SDK (Software Development Kit) applies to New CASIO Pocket Viewer (PV-S1600) only, no other models are supported.

2) System Requirements

OS:	Windows 98 Windows Me Windows 2000 or Windows XP
CPU:	A Pentium III 750MHz or later processor
Memory:	At least 128 MB (RAM) of available memory
Hard disk:	At least 30 MB of available hard disk space
Monitor:	A VGA monitor or higher

3) Before you use this SDK, you need to install the "File Transfer Manager" from CD-ROM of PV-S1600.

4) Install SDK to default directory "C:\CASIO\PV3XXX".

5) It is prohibited to make a copy or redistribute any part of this SDK, except that you may make one copy of the software solely for backup.

6) In no event shall CASIO (CASIO Computer Co., Ltd.) be liable for any damages whatsoever arising out of the use or inability to use the SDK software.

7) The contents of this Software Development Kit can be changed without notice to improve it.

8) Windows is registered trademarks of Microsoft Corporation.

9) Pentium is a registered trademark of Intel Corporation.

10) Other unspecified names in this SDK such as CPU, device and product names are either registered trademarks or trademarks of respective developers.

2. Overview

1) Hardware

CPU	: SH-3(by HITACHI). 29.5MIPS OSC;14.744MHz Core/Bus Clock;29.488MHz
MEMORY	: FLASH 16MB (OS & Application and User data). SD-RAM 8MB (for work (Program and data)).
D/D	: CASIO original.
LCD	: 160 x 160 mono-chrome Full dots with EL Back light.

Touch Panel controller.
Real Time Clock.
USB 1.1

2) Software

FLASH	: OS & Built-in Application 4MB Add in software and User data 12MB
OS	: CASIO original New OS for PV-S1600.(Application uses only library)
Program size	: Depend on free size of SD-RAM
RAM size	: Depend on free size of SD-RAM

3) Tool

C Compiler	: Hitachi C Compiler (by HITACHI)
PC simulator	: CASIO original

4) Documents

SDK_MAN.pdf.	:This file.
Lib_Func_Man.pdf.	:The detail document of library function

3. Install & Directory

3-1. Install

Install SDK to default directory "C:\CASIO\PV3XXX". The settings of the following files that the installation directory path of this environment is "C:\CASIO\PV3xxx". And if you change the installation directory, you need to change the setting of C compiler directory "C:\pvshcom" in the following files.

<CASIO\PV3xxxx>
PathSet.bat
shcdir.def

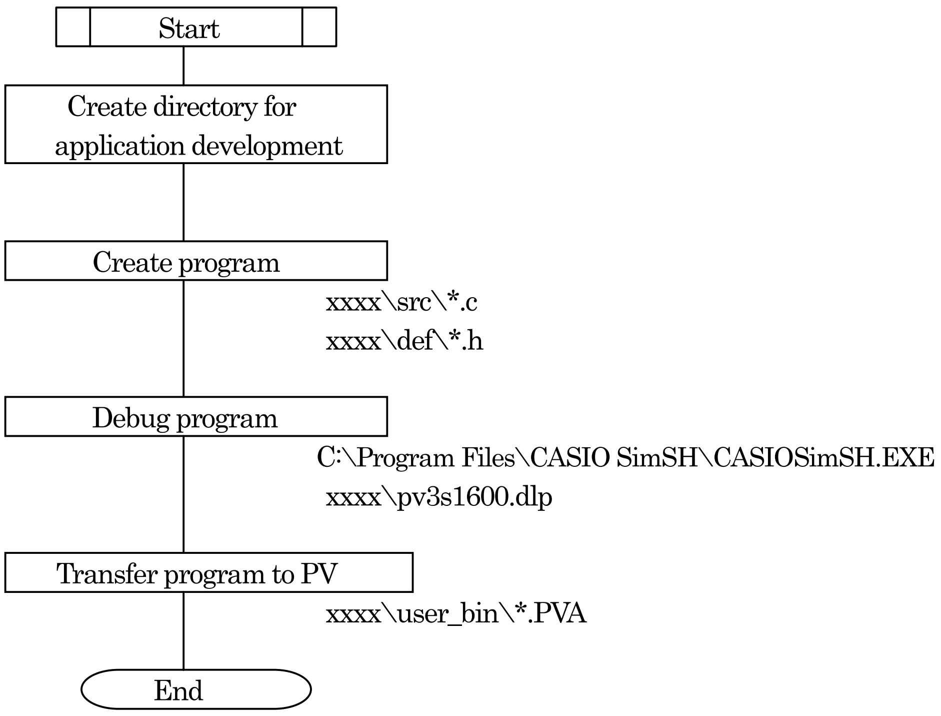
3-2. Directory Structure

```
C:\CASIO\<PV3xxxxx\>          ('xxxx' is model name of SDK)
   │   PATHSET.BAT                     Batch file for Path setting
   │   SHCDIR.DEF                      Directory setting file for C compiler
   │   License.txt                     License Agreement
   ├────── <Sample\>          <Sample Program for development add-in software>
   ├────── <TextEdit\>        <Sample Program for accessing the binary file>
   ├────── <Scan\>            <Sample Program for Event Library>
   ├────── <DOC\>             <Document files>
   │        SDK_MAN.pdf                This file
   │        Lib_Func_Man.pdf           The document of library function
   ├────── <LIB\>             <PV library>
   │        │LibC.lib                  Library file
   │        │AddLibC.lib               Additional Library file
   │        ├────── <LibH\>            <Header for Library>
   │        └────── <HiLIB\>           <Header for Porting tool>
   ├────── <SYSTEM\>          <SYSTEM>
   │        │
   │        └────── <OBJ>              <Start up Object>
   ├────── <Tools\>           <Tool>
   │        BuildAll.BAT               Batch file for Make execution
   │        makefile.sub               Sub command file for Release Makefile
   │        makedbg.sub                Sub command file for Debug Makefile
   │        sethead.exe                Tool to add the header in PVA
   │        putname.exe                Tool to set the application name
   │        seterr.exe                 Tool for batch operation
   │        genscr.exe                 Supporting tool to make the script
   └────── <Sim\>             <PV Simulator>
            │  pvs1600.dlm             Model file for simulator
            │  KeyPV1600.bmp           Graphic file for key board window
            │  KeyPV1600.txt           Setting file for key board window
            │  LcdPV1600.bmp           Graphic file for PV background
            ├────── <SETUP\>           <Simulator Setup Files>
            ├────── <INIT\>            <Memory Initial Data>
            └────── <MEM\>             <Memory Saving Data>
```

```
C:\PVSHCOM
  <SHC\>              <C compiler>
    <BIN\>                  <Compiler Program>
    <INCLUDE\>              <Header>
    SHCLIB.LIB              Standard Library
  <MANUALS\>                <Manual>
    SHCMAN.pdf              Hitachi C compiler & Linker Manual
    ESHCRN13.pdf            Limitation for Assembler
```

4. Outline of Development Procedure

Development of a new PocketViewer add-in program is performed in principle with the following procedure:



4-1. Outline of directory structure

```
C:\
<CASIO>
 │
 ├── <PV3xxxx>  AP
 │     │
 │     ├──── <SAMPLE>            Working directory for sample add-in files
 │     │      │ pv3s1600.dlp     Simulator Project file for sample program
 │     │      │ pv3s1600.dlr     Simulator Project file for execute information
 │     │      │ pv3s1600.dlw     Simulator Project file for window information
 │     │      │ BuildAll.bat     Make execution batch file for making PVA and abs
 │     │      │ sources.def      Information setting file for application
 │     │      │
 │     │      ├──── <SRC>        Source files for add-in programs (*.c)
 │     │      ├──── <DEF>        Source files for add-in programs (*.h)
 │     │      ├──── <ICON>       Icon source for add-in programs
 │     │      ├──── <debug>      Working directory for debug build
 │     │      ├──── <release>    Working directory for release build
 │     │      ├──── <make>       Default make script for building program
 │     │      └──── <USER_BIN>   Directory to which generated programs are output
 │     │                                 (*.abs,*.dbg,*.pva)
 │     │
 │     ├──── <DOC>               Document directory
 │     ├──── <SIM>               Directory of default setting files for simulator
 │     │      ├──── <SETUP>      Directory of Simulator Setup files
 │     │      ├──── <INIT>       Directory of OS binary files for simulator
 │     │      └──── <MEM>        Directory to save memory files for simulator
 │     └──── <TOOLS>             Tools
```

*** "xxxxx" will be replaced with a model name: "PV3S1600" for PV-S1600.**

4-2. Creating directory for development

1) Create a directory for development directly under the directory into which the SDK was installed (C:\CASIO\PV3xxxxx). There are no restrictions in particular on the directory name, except that it shall not be a long name. However, an understandable name should be used.

Example: <PV3xxxxx>

```
 │
 ├────      <SAMPLE>
 ├────      <DOC>
 └────      <GAME>  ◄──       New project directory
```

2) Copy all the contents of the SAMPLE directory to the directory you have just created.

Example:

```
<SAMPLE>                      <GAME>
 |                             |
 |  BuildAll.BAT     -->       |  BuildAll.BAT    Batch file for build
 |  sources.def      -->       |  sources.def     Definition file for application build
 |  pv3s1600.dlp     -->       |  pv3s1600.dlp    Project file for simulator
 |  pv3s1600.dlr     -->       |  pv3s1600.dlr    File to save execution state of simulator
 |  pv3s1600.dlw     -->       |  pv3s1600.dlw    File to save window state of simulator
 |---- <SRC>         -->       |---- <SRC>        Directory of C source files
 |---- <DEF>         -->       |---- <DEF>        Directory of C header files
 |---- <icon>        -->       |---- <icon>       Directory of bitmap images for menu icons
 |---- <debug>       -->       |---- <debug>      Working directory for debug build
 |---- <release>     -->       |---- <release>    Working directory for release build
 |---- <make>        -->       |---- <make>       Directory of default make scripts
 `---- <user_bin>    -->       `---- <user_bin>   Directory to which generated programs
                                                    are output (*.abs/dbg/pva)
```

4-3. Creating program

1) Creating program

When the directory is ready, actually create a program. Create C source codes in the SRC directory and header files in the DEF directory for the project. Create icon data for programs in the ICON directory as icon.bmp (menu) and licon.bmp (list). If you do not need icons, remove the BMP files in the ICON directory. In that case, the default icons prepared by the system will be used.

Example:

```
<GAME>
 |  BuildAll.bat
 |  sources.def
 |  pv3s1600.dlp
 |  pv3s1600.dlr
 |  pv3s1600.dlw
 |------ <src>
 |         puzzle.c
 |         gdata.c
 |------ <def>
 |         GAME.H
 |------ <icon>
 |         icon.bmp
 |         Licon.bmp
 |------ <DEBUG>
 |------ <RELEASE>
 |------ <Make>
 `------ <User_bin>
```

2) Rewriting sources.def

Modify sources.def, the definition file for compilation, according to the program you are going to create.

```
#######################################################################
#   Application Independent Definitions
#######################################################################

#######################################################################
#   Application                                                       #

# Create *.pva,*.abs,*.dbg file name
TARGET=GAME                                          Change (*1)

# PV Application Title (This name is displayed on PV Menu)
TITLE="Puzzle"                                       Change (*2)

# Program Version(EX. 0100->Ver1.00)
VERSION = 0100                                       Change (*3)

# Application Sources (*.c)
FILE0=puzzle                                         Change (*4)
FILE1=gdata                                          Change (*4)

#######################################################################
```

(*1) Specify the names of files that are generated. In case of this example, files GAME.ABS, GAME.DBG and GAME.PVA are generated under user_bin.
For a name you set here, use a string of alphabets and/or digits containing eight characters or less.

(*2) Specify the character string that is displayed as the program title in the menu screen.

(*3) Specify the version. This version is displayed when you touch [MenuBar]-[OPTION]-[VERSION] from the main menu of PocketViewer.

(*4) Set the source files to be compiled for variables that are defined in order from zero, such as FILE0, FILE1, FILE2 and so on.

3) Run BuildAll.BAT to perform build.
Execute "BuildAll.BAT". "BuildAll.BAT" makes the "Build.LOG" file as result.
If error or warning is occurred, the result is described in Build.LOG.
After build, check the Build.LOG file.
Reference the Compiler manual about error and warning.

4) When build is successful, three files are created in user_bin under the project directory, with the name specified as TARGET in sources.def and with extensions pva, dbg and abs. Abs and dbg files are used by the simulator. Pva files are program files that operate with actual PV devices as well as the simulator. You will eventually download *.pva files onto PV using the FTM that is described later.

Example: <user_bin\>

```
| GAME.ABS  ←   Created add-in program (for simulator)
| GAME.DBG  ←   Created add-in program (for simulator)
| GAME.PVA  ←   Created add-in program (for actual device/simulator)
|
```

4-4. Debugging program

1) Modify c:\casio\pv3s1600\game\pv3s1600.dlp as the following example:

Example:

```
 "PV3S1600.dlp"
        :
[Program1]
Program=user_bin\sample.abs            ←   Change "sample" to "GAME"
Debug=user_bin\sample.dbg              ←   Change "sample" to "GAME"
LoadAddress=00000000:00000000
[LoadInternal]
Line0=user_bin\sample.pva              ←   Change "sample" to "GAME"
    :
```

2) Run the simulator.

3) [Operation on simulator]
Click **[Project]-[Open]** from the main menu of the simulator, and then select and open pv3s1600.dlp in the directory where the project is located (c:\casio\pv3s1600\game for this example).

4) [Operation on simulator]
If necessary, execute **[Project]-[Clear Memory and Reload]**. Doing this will clear all the contents of the memory. If this is not done, you will perform debugging with the memory in the same condition as that when the simulator last exited.

5) [Operation on simulator]

Execute **[Run]-[Run]** from the main menu of the simulator. This will start simulation and the reset window will appear like an actual device. Then, perform reset operation as with usual PV before entering the menu window.

6) [Operation of PV during simulation]

After the menu appears, touch the APound or Ao button displayed on the right of the window to switch the menu. This draws the title and icon of a program to be debugged. Touching the title or icon will start the program to be debugged.

4-5. Downloading add-in program into actual device

1) Connect PV with your PC using a serial or USB cradle cable. After that, start the File Transfer Manager (FTM.EXE).

2) [Operation with File Transfer Manager (FTM)]

Click **[Casio]-[Options]** and set the type of the cradle cable used for connection and, if it is a serial connection, the number of serial port used for connection.

3) [Operation on PV]

After clicking on the menu icon, click **[MenuBar]-[CommunicationSetup]** and set the type of the cradle cable used for connection with the PC.

4) [Operation on PV]

Again from the main menu, click **[MenuBar]-[TransferFile]**. This brings PV in a wait state for connection.

5) [Operation with File Transfer Manager (FTM)]

Click **[Casio]**, and then click **[Connect to Casio]**. This brings PV and your PC in a link state. And when you click Applications in the right side window that displays PV's condition, the FTM will display conditions of application programs that are installed in PV.
After that, move the position of directory displayed in the left side window to the directory where PVA files are located (c:\casio\pv3s1600\game\user_bin in case of the example). And drag and drop the PVA files there onto the right side window that displays the condition on PV's side.

Refer to the manual for FTM for more detailed operation method.

4-6. Porting previous PV software versions to new PV

This SDK provides a program called Porting Tool to port previous PV software versions. By using this tool, typical conversions necessary for porting can be automatically performed. This helps you to port previous PV software versions to new PV.

Refer to the instruction manual of Porting Tool for details.

4-7. More advanced debugging

The simulator enables you to send data files on the Windows side into PV that is performing simulation.

1) Specifying file to be transferred
Select **[Project]-[Edit]-[Load Internally]** and specify files to be transferred, with the first one for Read filename 1, second one Read filename 2, and so on. For Read filename 0, you must specify in principle a pva file of the program to be debugged.

2) Type of file that can be transferred
Files that can be put in are generic binary files or AID files. These files are generally transferred into the user data area of PV.

3) Note
The files you specified are transferred to PV that performs simulation, after a reset operation of PV is complete. However, a file is not written into PV if a file with the same name exists on PV (it is not overwritten).

For this reason, conditions of the last session can be maintained as long as you do not clear the contents of PV's memory when you reset it. If you want to start from the beginning in disregard of the conditions of the last session, execute **[Run]** again after performing **[Clear Memory and Reload]** from the simulator to clear completely the flash memory before you start debugging.

5. Application Design Restrictions

Code restrictions

1) Code size that can be described is dependent on the remaining memory quantity of the system. Generally, design can be performed in a size within which you can use about 2Mbyte memory space, which will contain all codes including program codes and un-initialized variables.

In the standard environment provided by the SDK, you can design and debug a program up to the following sizes. However, please note that a program that has a larger size will take longer to start up.

Program code (P) + Data that cannot be rewritten (C) + Un-initialized data (B) = 2MB
Initialized data (D) = 128KB
Heap = 32kB

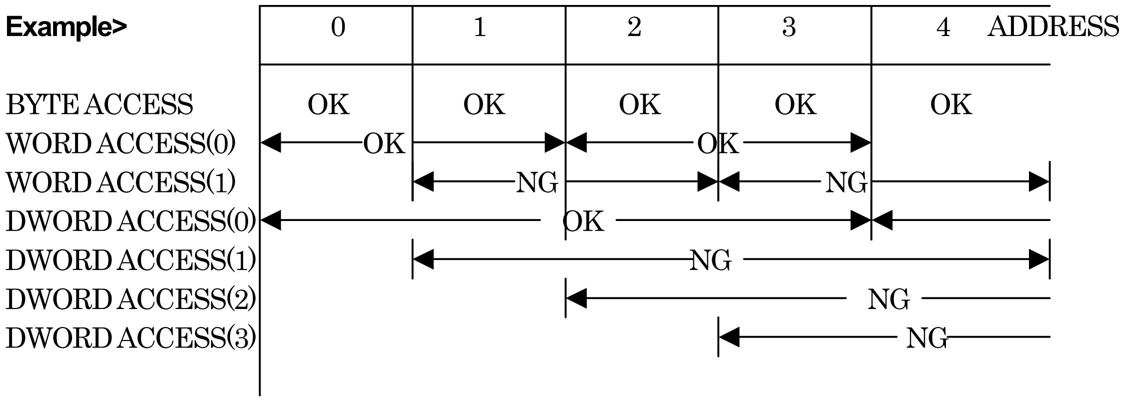
2) Operation of the standard libraries is guaranteed only for functions that are defined by HITACHI.H. Other libraries should be used on your own responsibility.

3) This system uses Hitachi's 32-bit RISC CPU SH-3 for the CPU. For this reason, memory access must be done from a boundary that has the same number of bits as access bits, unlike with previous PocketViewer products.

For example, you can only access addresses that are divisible by 2 when accessing in 2-byte units, and addresses that are divisible by 4 when accessing in 4-byte units.

Be careful because this fact is likely to be overlooked especially when accessing different sized data directly by using a pointer that has a different access size, or accessing data inside a structure.

Example>



4) When creating a structure, if structure members have different sizes, the compiler may automatically insert dummy data between the members to avoid the problem described in 3) above.

Take care when performing an operation that is influenced by the actual size of a structure, for example, when accessing a data file using a structure.

```
Example>      Definition                 Actual memory arrangement
              struct _A{               struct _AA  {
                byte b;                    byte b;
                word w;                      byte dummy1; // Inserted by compiler
                byte b2;       →           word w;
               dword dw;                     byte b2;
              }A;                          byte dummy2[3]; // Inserted by compiler
                                           dword dw;
                                         }AA;

              sizeof(A)         =        sizeof(AA)
```

5) Memory arrangement is in a big endian arrangement, unlike with previous PocketViewer products. For this reason, the memory image used when storing a dword (long) type or word (short) type into memory has been changed. Therefore, care must be taken when porting a program dependent on a memory image from previous PV to new PV.

Example>

Previous PocketViewer (Little Endian)

Address	0	1	2	3
	0x78	0x56	0x34	0x12
	0x12345678			

This PocketViewer (Big Endian)

Address	0	1	2	3
	0x12	0x34	0x56	0x78
	0x12345678			

6) Never rewrite a variable that was declared as const, because that operation will cause the system to be forcefully stopped in order to protect the memory.

7) A general library function includes the character string "Lib" at the head of its function name.

8) In the PVOS, the amount of stack that is allocated to applications is very small, only about 4 Kbytes. For this reason, do not use in principle a local variable that has a large sized array.

In particular, do not design an application using an algorithm that consumes a large amount of stack, such as a recursive algorithm.

If you use a large sized array, you should use a global variable, which usually is not initialized.

9) Be sure to use the following type definitions:

```
typedef unsigned char    byte;
typedef unsigned short   word;
typedef unsigned long    dword;
```

10) Because new PV uses 32-bit CPU SH-3 for the CPU, the size of an int type is 32 bit, unlike with previous PocketViewer products. When handling data that has a size of 16 bits, use a short type or word type.

11) Do not use a bit field.

12) When calling a function that you created and will open to the public, be sure to include the header that was used to perform the prototype declaration before calling that function.
Because the compiler used in new PV does not check a type strictly when it calls one, it does not output a warning even if the argument type is not the default type(int), when the prototype declaration is not performed.

13) Use CASIO libraries for calculation of real numbers.

14) The BIOS or a variable in a library cannot be referred to or rewritten because applications are separately linked. Also, never rewrite a value that is dependent on an address because a library is placed on an undefined address.

15) Do not use a long filename. Some tools do not support it.

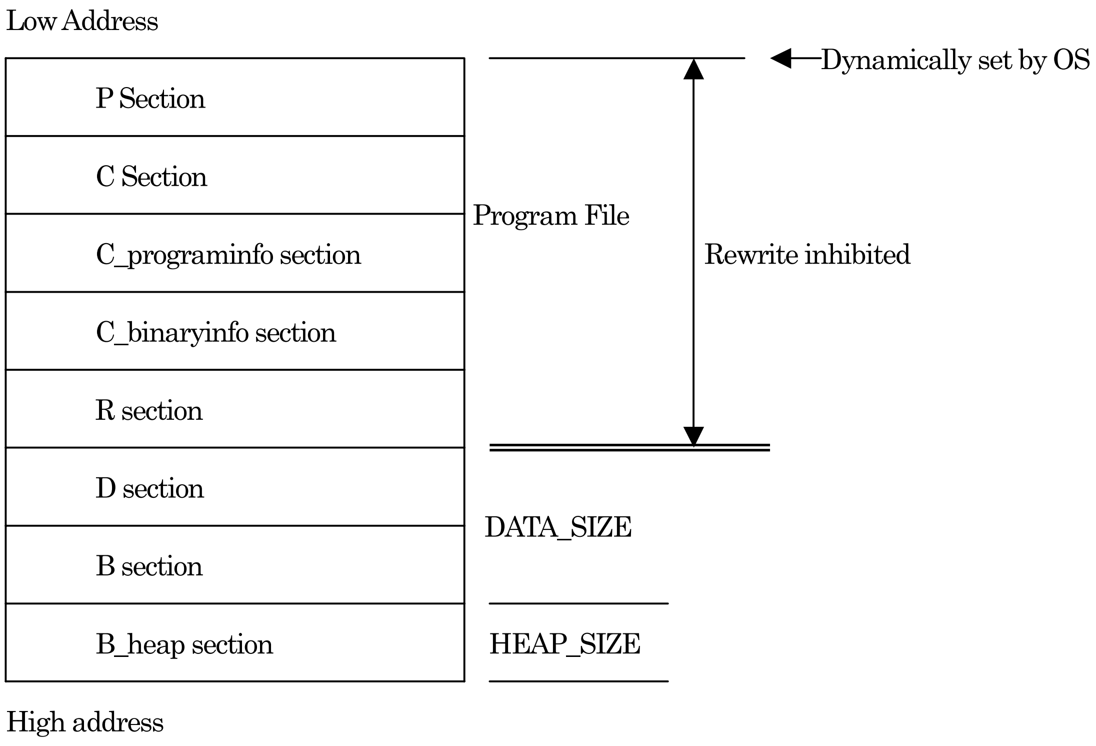
16) You cannot use an option that is optimized using a GBR register, such as #pragma gbr_base or #pragma gbr_base1.

6. PV Memory Map

6-1. Image within application process

When the application is tapped from the menu, the system reads out the program file and expands it to the memory. At this time, memory is automatically allocated in the following section structure. Addresses of allocated memory are not fixed to specific addresses, because they are dynamically managed by the memory manager.

Addresses allocated in general applications are between 0x10000000 and 0x5FFFFFFF.



Details:

P section = Program code
C section = Constant data (const)
C_programinfo section = Program managing information
C_binaryinfo section = Process managing information
R section = Initialization data of data with initialization (copy of D section)
The contents of the R section are copied to the D section when the process is restarted.
D section = Data area with initialization
B section = Data area without initialization
B_heap section = Heap area

6-2. Memory image throughout PocketViewer

	Address	
	$00000000	
Reserved		
	$00001000	
BIOS CALL TABLE		
	$00001800	
Reserved		
	$00010000	
Application		
	$60000000	
Normal Driver		
	$70000000	
Shared memory		
	$80000000	
OS (BIOS&WORK)		
	$A0000000	
IPL (MASK ROM)		
	$A0010000	
Reserved		OS (BIOS/WORK mirror area)
	$C0000000	
Reserved		(Area used at constant addresses for special use)
	$D0000000	
Low Level Driver		
	$E0000000	
I/O		
	$FFFFFFFF	

*** Access to memory from $80000000 onward is prohibited.**

7. SIMULATOR

7-1. Overview

Welcome to CASIO SimSH Simulator! This program is used in program development on CASIO's new range of PV handheld computers.
This program simulates the function of the PV unit, and emulates the PV hardware, including the CPU, which makes it possible to debug programs on a PC without any special hardware.
During debugging, you can look at the source code and disassembled code to be executed, and check variable contents, registers of the CPU, and the memory areas of the PV unit.
Each program to debug is controlled through a project file. This makes it easy to switch between programs.
It is also possible to debug programs to be run on different PV models. The project file includes the type of PV model being used. The model file describes the hardware in the PV unit.

7-2. System Requirements

The execution speed of the simulator is proportional to the CPU and memory speed on the computer being used. The following list is the recommended system to use for simulating a PV unit.

OS	Windows 98/Me, Windows NT/2000/XP
CPU	Pentium III 750 MHz or higher
Memory	128 MByte free
Hard disk	Free space of 30 Mbyte

7-3. Limitations

The simulator has a few limitations from the actual hardware CPU. These limitations are the following.

- The instruction and data caches are not used. The cache registers are accessible, though.
- The execution time clock counting in the simulator is different from the actual CPU. The actual execution time of an instruction in the PV unit varies, e.g. by cache hit rates and memory speed. In the simulator, the execution time clock increases by an average instruction execution time instead.

7-4. Using the simulator

This chapter explains how to use the SimSH simulator to debug a program.

Starting

When the simulator has been started, the default project and setup of windows is displayed. All windows are usually empty, since the default project include an empty model (without any device drivers).

Creating a new project

You need a project file to inform the simulator where to find and place the files for the program. Create a new project, from menu **Project**, **New**. (If you prefer, you can reuse the settings from an old project, from menu **Project**, **Open** or **Project**, **Recent projects**.)

Setting up the model for the project

You also need to inform the simulator which PV unit the program is to be run on.
Use menu **Model**, **Open** to find the right PV unit model. (If you have selected this model earlier, you can also find it on the recent list with menu **Model**, **Recent models**.)Setting up the project
Edit the project settings, from menu **Project**, **Edit**. This will open the *Edit Project* dialog. Press **OK** when the settings have been entered.

Saving the settings

After making these settings, save the project file with menu **Project**, **Save as**. You can select the location and file name of the project file.
From now on, you can select this project simply by opening the project file from menu **Project**, **Open** or **Project**, **Recent projects**.

Compile and link the program

When the source code of your program has changed, you need to compile and link the program before the new program can be debugged. The compilation and linking is performed by menu **Project**, **Build**.
To be able to debug the program after building it, it must first be loaded into the simulator. Do this from menu **Project**, **Reload**. If you forget this, the simulator will still inform you when it thinks you should reload the project.

Setting breakpoints

Breakpoints can be set in the program, to stop program execution when that part of the program is reached.
To set breakpoints in a specific source file, open that source code from menu **File**, **Open Module**.
Use the **Breakpoints** menu to set and clear breakpoints.

Debugging the program

Program function can now be checked from the **Run** menu. It is possible to single step the program, run to next call or return, and to run the program in full speed. The program will work in the same way as when run in the actual PV unit.
When single stepping the program, the behaviour is different when the *Source code* window and the *Disassembled code* window is the active one. In the *Disassembled code* window, one instruction is executed. In the *Source code* window, a number of instructions are executed, until another source code line is reached.
Use menu **Run**, **Trace into** (shortcut key **F8**) to single step the program, and go into each function called.
Use menu **Run**, **Step over** (shortcut key **Shift+F8**) to single step the program, and to step over each function called.
Use menu **Run**, **Run** (shortcut key **F5**) to run the program at full speed. Program execution will stop when a breakpoint is reached.
Use menu **Run**, **Animate** to run the program in automatic single step mode. The windows contents are updated after each instruction is executed. Program execution will stop when a breakpoint is reached.
Note that it is not possible to debug a program before the project file has been given a name. When the project has no name, the **Run** menu is disabled.

Stopping program execution

Program execution stops when a breakpoint is reached. If the program is not working correctly, it may not reach the breakpoint. Therefore it is possible to stop program execution manually.
Use menu **Run**, **Stop** (shortcut key **Shift+F5**) to stop program execution.

7-5. Program debugging is easy

Start up SimSH simulator

|

Open the program project. (**Project, Open**)

|

Change breakpoints if needed.
Open C source file. (**File, Open module**)
Set breakpoints. (**Breakpoints, Set**)

|

Run the simulated program. (**Run, Run**)

|

The program stops when a breakpoint is reached.
Stop the simulated program. (**Run, Stop**)

|

Close the SimSH simulator. (**File, Exit**)

7-6. Using breakpoints

Breakpoints are used when program execution must be stopped at a certain code line. The user can then check variable and memory contents, and start single-stepping the program.
There are three different types of breakpoints.

- Code breakpoint, active when a code instruction is read and executed.
- Data breakpoint, active when specific memory addresses are read or written.
- Exception breakpoint, active when an exception or interrupt occurs.

All breakpoints are automatically saved when the project is closed. When the project is loaded, all saved breakpoints are restored.

Setting breakpoints

Breakpoints can be set in the *Source code*, *Disassembled code*, *Global variables*, *Local variables*, and *Memory windows*.
Place the cursor at the line or memory location where the breakpoint is needed.
Use menu **Breakpoints**, **Set** to set the breakpoint. You can also use menu **Breakpoints**, **Toggle** (shortcut key **F9**) to toggle the breakpoint on and off.

Disable and Enable breakpoints

Some breakpoints might not be needed at all times. Such breakpoints can be disabled, so program execution does not stop at them. The breakpoint can later be enabled.
Use menu **Breakpoints**, **Disable** to disable the breakpoint, and menu **Breakpoints**, **Enable** to enable the breakpoint.
There is a symbol in the left border area of the *Source code*, *Disassembled code*, *Global variables*, *Local variables*, and *Memory windows*, that shows whether the breakpoint is enabled or disabled.

●	Breakpoint is enabled
○	Breakpoint is disabled

Editing breakpoints

Each breakpoint can be set with more conditions. These conditions are set from the *Edit breakpoints* dialog. The following conditions can be set.

Status	The status of the breakpoint. This is explained below.
Access	The memory access for when the breakpoint is active. This is explained below.
Breakpoint wait count	The wait count to use for the breakpoint. The wait count is decremented each time the breakpoint is reached. Program execution stops at the breakpoint when the count reaches 0.

Status condition

The following status conditions can be set for the breakpoint.

Enable	The breakpoint is enabled.
Disable	The breakpoint is disabled.
Hide	The breakpoint has been cleared. All conditions of this breakpoint is kept, so the user can enable this breakpoint if it was cleared by mistake.
Delete	The breakpoint is to be deleted, when the dialog is closed.

Access condition

The following memory access conditions can be set for the breakpoint.

Code read	The breakpoint is active when the code is read for program execution.
Data read	The breakpoint is active when the memory is read for data access.
Data write	The breakpoint is active when the memory is written to by data access.
After access	When this condition flag is off, program execution will stop before the memory is read or written by the data access. This is useful when the previous memory contents needs to be checked before being written to. When this condition flag is set, program execution stops after the memory read or write has been performed.

Exception breakpoints

A breakpoint can also be active when an exception occurs. An exception is a special condition in the processor that stops normal program execution. Some types of exception are: interrupts, reset inputs, and memory errors.. The following action can be set for exceptions.

Continue execution	Continue execution of the exception and the normal program code.
Stop execution	Stop execution, and display a message in the status line describing the exception.
Show an error	Stop execution, and display a message in a dialog box describing the exception. The user must accept the dialog box.
Use default handling	Use the default handling of the exception. The default handling is to only stop execution and show an error if something looks to have gone wrong.

7-7. File

7-7-1. File menu

The File menu is used for open source files, and to save all changed files at once.
The following entries are available in the File Menu.

Open module	Open another file module in the program being debugged. The *Open Module* dialog box is displayed, where a file module can be selected.
Save LCD pixels	Save the LCD pixel contents to a bitmap file. The file is saved in monochrome or colour mode, depending on the settings in the *LCD and Touch panel Properties* dialog box.
Save all	Save all changed settings and files. This includes the model file, project file, and window positions and properties.
Exit	Exit the simulator. If there are settings or files that are changed but not saved, the program asks before saving them.

7-6. Edit

7-6-1. Edit menu

The Edit menu handles moving the cursor location in the windows, and editing memory and variables.
The following entries are available in the Edit Menu.

Goto	Display the *Goto* dialog box, to make it possible to go to another location in the current window.
Modify	Display the *Edit* dialog box, to change the value of the selected variable or memory location.
Show memory	Display the current location at the cursor in the *Source code*, *Global variables*, or *Local variables* window, as the corresponding address in the *Disassembled code* or *Memory window*.
Properties	Display the *Properties* dialog box, to see and change how the window contents are displayed.

7-7. Project

Each program to debug is controlled through a project file. The project file contains all information needed, such as PV model, program file, and debug information to be used.

7-7-1. Project menu

The Project menu handles loading, saving, and editing of projects.
The following entries are available in the Project Menu.

New	Clear the project settings to the default values.
Open	Open a project file.
Save	Save the current project file with the current file name.
Save as	Save the current project file with a new name.
Recent projects	Open a project file, from a list of the most recently used projects.
Build	Build the project to create a new program file.
Edit	Edit the project file to change the settings of the project. The *Edit Project* dialog box is opened.
Reload	Reload the project file, including the model settings, device drivers, and the program file. The memory files will not be saved. The files saved last time will be used, which is useful if the memory contents have been destroyed during debugging.
Save memory, reload	Save the memory files with the current memory contents. Then reload the project file, as for the **Reload** command.
Clear memory, reload	Clear the memory files used by this project. Then reload the project file, as for the **Reload** command.
Load memory	Load the memory files last saved for this project.
Save memory	Save the memory files with the current memory contents.
Load program	Select a new program file to load into the simulator. Any program files and debug information already loaded will still be available in the simulator. Program files and debug information loaded with this command will be discarded when a new project is loaded, or the current project is reloaded.

7-7-2. Edit Project dialog

This dialog is accessed through menu **Project**, **Edit**.
In this dialog all project settings are entered, except for the model. The model to use must be selected from the Model menu.
The following fields and tabs can be set for the project.

Project name	The name of the project, to display in the title bar.
Memory path	The path where the modified memory files are to be saved and loaded. Press the button on the right of the input field, to browse for the memory path.
Source path	The path where the source files of the program can be found. More than one path can be entered. Press the button on the right of the input field, to browse for a new source path. Usually the full path of the source files are stored in the debug information file. Still it is useful to set the root entry of the project source files as the source path. This makes it possible for the simulator to find the source files if the project files (including the source files) are moved to another drive or path.

Load 1-5

Up to 5 different program and debug files can be entered in the project.
The following fields can be set for this tab.

Program file	The file of the built program to be debugged for this project. Press the button on the right of the input field, to browse for the program file.
Debug info file	The file of the debug information for the program to be debugged. Press the button on the right of the input field, to browse for the debug information file. The debug information can also be included in the program file, and no separate debug information file is created. In such cases leave this field empty.
Load address	The address where to load a binary file. Enter the address as a hexadecimal number. This value is ignored for an executable file that includes the load address in the program file.
Load mem mode	The memory mode that the program will run in. This entry is used with the debug information. To ignore the memory mode for the debug information, set this field to **Current mode**.
Load mem type	The type of memory access that the program will run in. This entry is used with the debug information. To ignore the memory access type for the debug information, set this field to **Current memory type**.

Load internally

The PV unit can also load files internally from the OS. The file names to use are entered at this tab. The tab is displayed only if the current PV model supports this function.
The following fields can be set for this tab.

Edit file name	The file name or path to use for the setting selected in the list. Press the button on the right of the input field, to browse for the file name or path.

7-8. Model

Each PV unit has a corresponding model file. The model file describes the hardware of the PV unit; which devices are used within the PV unit, and how the devices are connected.

7-8-1. Model menu

The Model menu handles loading, saving, and editing of models.
The following entries are available in the Model Menu.

New	Clear the model settings to the default values.
Open	Open a model file.
Save	Save the current model file with the current file name.
Save as	Save the current model file with a new name.
Recent models	Open a model file, from a list of the most recently used models.
Edit	Edit the model file to change the settings of the model. The *Edit Model* dialog box is opened.

7-8-2. Edit Model dialog

This dialog is accessed through menu **Model**, **Edit**.
In this dialog all model settings are entered. The model holds information about the PV model being simulated.
The following fields can be changed for the model.

Device	The drivers and devices used in the model. Only the drivers and devices that have a control setting are displayed in the tree. The devices for a driver is displayed by pressing the + sign in front of the driver name.
Control setting	The control settings that can be set for the currently selected device. Select a device in the device tree, to see the control settings for that device.
Edit	The current value or string for the selected control setting. You can enter the new value or string. Enter an empty string to remove this setting from the model, and to use the default setting. The previously entered values are saved in the drop-down list.

7-9. Run

The program is executed and debugged by emulating the processor instructions and simulating the hardware of the PV unit.

7-9-1. Run menu

The Run menu handles program debugging within the simulator.
The following entries are available in the Run Menu.

Stop	Stop program execution. This command is used if program execution continues without reaching the expected break points.
Continue	Continue program execution using the last run command. This command keeps any temporary breakpoints set by the **Step over** and **Run here** commands.
End	End program execution. After this command is used, the project needs to be reloaded.
Run	Run the program in the simulator, up to the next breakpoint.
Trace into	Step program execution line by line. Any functions being called will also be stepped through line by line.
Step over	Step program execution line by line. Any functions being called will be run without stepping.
Run here	Run the program until the line where the cursor is located would be executed.
Run to call/return	Run the program until the next function call or return is executed.
Run to return	Run the program until the execution has returned from the current function.
Animate	Step program execution automatically line by line, updating the windows after each line. Stop execution with the **Stop** command.
Trace all instructions	Step program execution line by line. All functions being called, including interrupts and exceptions, will also be stepped through line by line.
Show next instruction	Show the next instruction to be executed in the *Source code* and *Disassembled code* windows. This command is useful after using the *Open Module* or *Goto* dialog.
Set next instruction	Change the instruction pointer to another program line. This command is useful to repeat or skip some code lines due to a discovered bug in the program being debugged. Make sure the stack pointer and other registers are also updated if needed, when the program execution location is changed.
Save to log file	When this entry is checked, all changes to the instruction pointer are saved in a log file. The log file has the same name and location as the project file, but with extension dlg. New entries are added to the end of the log file.

Note that it is not possible to debug a program before the project file has been given a name. When the project has no name, the Run menu is disabled.

7-10. Breakpoints

Breakpoints can be set to stop program execution when a certain code instruction is read, or when memory is read or written.

7-10.1. Breakpoint menu

The Breakpoints menu handles setting and editing of breakpoints for program execution.
The following entries are available in the Breakpoints Menu.

Set	Set a breakpoint on the line at the cursor.
Clear	Clear the breakpoint on the line at the cursor.
Toggle	Toggle the breakpoint on the line at the cursor.
Enable	Enable the breakpoint on the line at the cursor. Enabled breakpoints are displayed as a filled circle.
Disable	Disable the breakpoint on the line at the cursor. Disabled breakpoints are displayed as an unfilled circle.
Enable all	Enable all breakpoints currently set.
Disable all	Disable all breakpoints currently set.
Edit	Edit the breakpoints. The *Edit Breakpoints* dialog box is opened.

7-10-2. Edit breakpoints dialog

This dialog is accessed through menu **Breakpoints**, **Edit**.
In this dialog all breakpoints are displayed, and can be edited. Also the access type to use when new breakpoints are set can be edited. The following tabs can be selected.

Breakpoints	The normal breakpoints currently enabled, disabled, or hidden are displayed.
Exceptions	The exception breakpoints are displayed.
Settings	The access type to use for new breakpoints are displayed.

Breakpoints

In this tab, each breakpoint is displayed with its status, location, type of access, size, and wait count.

Symbols

The status of the breakpoint is displayed with a symbol. The following symbols are available.

●	Breakpoint is enabled
○	Breakpoint is disabled
✖●	Breakpoint is hidden
✖	Breakpoint will be deleted when **OK** or **Apply** button is clicked

Buttons

The following buttons can be used for the breakpoints.

Edit breakpoint	Press this button to edit the breakpoint settings. The *Edit one breakpoint* dialog will be opened.
Goto breakpoint	Press this button to view the breakpoint location in the *Source*, *Disassembler*, or *Memory* window. If the type of access for the breakpoint is set to code read, the *Source* or *Disassembler* window will be used. Otherwise the *Memory* window is used. If any breakpoint has been edited, this button is disabled. You need to **Apply** the new settings first, so the changes will not be lost.
Restart count	Press this button to restart the wait counter to the initial value.
Delete all hidden	Press this button to mark all hidden breakpoints as deleted. The deleted breakpoints will be removed when the **OK** or **Apply** button is pressed.

Exceptions

Each exception is displayed with its name, current setting, and default setting.
To change the settings, mark one or more exceptions, and select the event to happen when that exception occurs.

Symbols

The status of the exception is displayed with a symbol. The following symbols are available.

●	Stop execution at exception
○	Continue execution at exception
❗	Show an error at exception
●	Stop execution at exception (this is the default handling)
○	Continue execution at exception (this is the default handling)
❗	Show an error at exception (this is the default handling)

Settings

The access type can be selected differently for *Source code*, *Disassembled code*, *Variables* (*Global* and *Local*), and *Memory* windows.
When the **Ask each time** box is checked, the *Set breakpoint* dialog box will be displayed each time a new breakpoint is to be set. The access type can then be selected for that breakpoint.

7-10-3. Edit one breakpoint dialog

This dialog is accessed by pressing the **Edit breakpoint** button in the *Edit breakpoints* dialog.
You can edit the breakpoint settings.
The following fields can be edited for the breakpoint.

Address	The starting address for the breakpoint. Enter the address as a hexadecimal number. This field cannot be edited if the breakpoint is set from the *Source* window.
Number of bytes	The number of bytes where the breakpoint is active, counted from the starting address. Enter the size as a hexadecimal number.
Mode	The memory mode for the breakpoint address. To ignore the memory mode for the breakpoint, set this field to **Current mode**. This field cannot be edited if the breakpoint is set from the *Source* window.
Type	The type of memory access for the breakpoint address. To ignore the memory access type for the breakpoint, set this field to **Current memory type**. This field cannot be edited if the breakpoint is set from the *Source* window.
Status	The status of the breakpoint.
Access	The memory access for when the breakpoint is active.
Breakpoint wait count	The wait count to use for the breakpoint. The wait count is decremented each time the breakpoint is reached, and the program execution stops when the count reaches 0. The initial count is the count set from the beginning. This count is used when the project is reloaded, or when the **Restart** count button is pressed in the *Edit breakpoints* dialog.

7-10-4. Set breakpoint dialog

This dialog is accessed through menu **Breakpoints**, **Set**, when the **Ask each time** box is checked at the tab **Settings** in the *Edit breakpoints* dialog.
You can select the access type to use for the breakpoint being set.
The following fields can be set for the access type.

Code read	The breakpoint is active when the code is read for program execution.
Data read	The breakpoint is active when the memory is read for data access.
Data write	The breakpoint is active when the memory is written to by data access.
After access	The breakpoint is activated after the memory has been accessed and updated.

7-11. View

There are a number of different window types available, to show source code and disassembled code, register, variable, and memory contents, and the screen of the PV unit.

7-11-1. View menu

The View menu handles opening of different types of windows in the simulator.
The following entries are available in the View Menu.

Source code	Display the *Source code* window.
Disassembled code	Display the *Disassembled code* window.
LCD and Touch panel	Display the *LCD and Touch panel* window.
Keyboard	Display the *Keyboard* window.
Execution status	Display the *Execution status* window.
Global variables	Display the *Global variables* window.
Local variables	Display the *Local variables* window.
Registers	Display the *Registers* window.
Memory	Display the *Memory* window.

7-12. Window

7-12-1. Window menu

The Window menu handles the windows currently displayed in the simulator.
The following entries are available in the Window Menu.

Duplicate	Duplicate the current window.
Close	Close the current window.
Close all	Close all windows.
Tile	Tile the windows so they are not hidden.
Cascade	Cascade the windows so they are on top of each other.
Arrange icons	Arrange the window icons to the bottom of the simulator window.
Edit	Set the Edit dialog as the topmost window.
1 - 9	Set the specified window as the topmost window.
More windows	Display the *Select Window* dialog box, to select or close another window.

7-12-2. Source code window

This window shows the source code of the program being debugged.
Breakpoints can be set at each line, from the **Breakpoints** menu.
It is also possible to go to another line in the file, a function, or another source file, from the *Goto line* dialog (with menu **Edit**, **Goto**) and the *Open module* dialog (with menu **File**, **Open module**).

Symbols

The symbols in the left border area of the window show the following information.

▷	This is the next line to be executed (the current PC location)
●	Breakpoint is enabled at this line
○	Breakpoint is disabled at this line

Keys

The following keys can be used in this window.

↑	Move the cursor to the previous line
↓	Move the cursor to the next line
←	Move the cursor to the previous character
→	Move the cursor to the next character
Page Up	Move the cursor to the previous page
Page Down	Move the cursor to the next page
Home	Move the cursor to the first character on the line
End	Move the cursor to the last character on the line
Ctrl+↑	Scroll the window upwards
Ctrl+↓	Scroll the window downwards
Ctrl+Home	Move the cursor to the first character in the source file
Ctrl+End	Move the cursor to the last character in the source file

7-12-3. Disassembled code window

This window shows the machine code of the program being debug, as disassembled code.
Breakpoints can be set at each line, from the **Breakpoints** menu.
It is also possible to go to the code of a function, or any other memory address, from the *Goto address* dialog with the menu **Edit**, **Goto**.

Symbols

The symbols in the left border area of the window show the following information.

▷	This is the next instruction to be executed (the current PC location)
●	Breakpoint is enabled at this address
○	Breakpoint is disabled at this address

Keys

The following keys can be used in this window.

↑	Move the cursor to the previous instruction
↓	Move the cursor to the next instruction
Page Up	Move the cursor to the previous page
Page Down	Move the cursor to the next page
Ctrl+↑	Scroll the window upwards
Ctrl+↓	Scroll the window downwards
Ctrl+Home	Move the cursor to the first instruction in this memory area
Ctrl+End	Move the cursor to the last instruction in this memory area

7-12-4. Registers window

This window shows the contents of all registers in the CPU.
The value of a register can be modified, from the *Edit* dialog with the menu **Edit**, **Modify**.
When the program is running, the window contents are not updated. The program must be stopped, or run in animation mode, to update the *Registers* window.
To avoid contention with program execution, it is only possible to modify the register values when the program is stopped.

Keys

The following keys can be used in this window.

↑	Move the cursor to the previous register
↓	Move the cursor to the next register
Page Up	Move the cursor to the previous page
Page Down	Move the cursor to the next page
Ctrl+↑	Scroll the window upwards
Ctrl+↓	Scroll the window downwards
Ctrl+Home	Move the cursor to the first register
Ctrl+End	Move the cursor to the last register

7-12-5. Variables window

This window shows the global variables of the current source file, or the local variables of the current function.
The current source file is the file where the next instruction is to be executed. The current function is the function where the next instruction is to be executed.
The value of a variable can be modified, from the *Edit* dialog with menu **Edit**, **Modify**.
When the program is running, the window contents are not updated. The program must be stopped, or run in animation mode, to update the *Variables* window.
To avoid contention with program execution, it is only possible to modify the variable values when the program is stopped.

Symbols

The symbols in the left border area of the window show the following information.

●	Breakpoint is enabled at this line
○	Breakpoint is disabled at this line
⊞	This variable has elements that are displayed by clicking on this symbol
⊟	The displayed elements of this variable is hidden by clicking on this symbol

Keys

The following keys can be used in this window.

↑	Move the cursor to the previous variable line
↓	Move the cursor to the next variable line
←	Hide the displayed elements of the variable at the cursor line
→	Display the elements of the variable at the cursor line
Page Up	Move the cursor to the previous page
Page Down	Move the cursor to the next page
Ctrl+↑	Scroll the window upwards
Ctrl+↓	Scroll the window downwards
Ctrl+Home	Move the cursor to the first variable line in the window
Ctrl+End	Move the cursor to the last variable line in the window

7-13-6. Memory window

This window shows the memory contents of any area in the PV model.
The memory can be displayed as bytes, words, or longs. The display layout is set from the *Properties* dialog with menu **Edit**, **Properties**.
It is possible to go to the address of a variable, or another part of memory, from the *Goto address* dialog with menu **Edit**, **Goto**.
The value of a memory location can be modified, from the *Edit* dialog with menu **Edit**, **Modify**.
When the program is running, the window contents are not updated. The program must be stopped, or run in animation mode, to update the *Memory* window.
To avoid contention with program execution, it is only possible to modify the memory values when the program is stopped.

Symbols

The symbols in the left border area of the window show the following information.

●	Breakpoint is enabled at this line
○	Breakpoint is disabled at this line

Keys

The following keys can be used in this window.

↑	Move the cursor to the previous line
↓	Move the cursor to the next line
←	Move the cursor to the previous memory location
→	Move the cursor to the next memory location
Page Up	Move the cursor to the previous page
Page Down	Move the cursor to the next page
Home	Move the cursor to the first memory location on the line
End	Move the cursor to the last memory location on the line
Ctrl+↑	Scroll the window upwards
Ctrl+↓	Scroll the window downwards
Ctrl+Home	Move the cursor to the first memory location in this memory area
Ctrl+End	Move the cursor to the last memory location in this memory area

7-12-7. Status window

This window shows the status of execution. This includes the next memory access to be performed, the current execution time, and any errors or pending interrupts and exceptions.
Part of this information is also displayed in the status line.

7-12-8. LCD and Touch panel window

This window shows the LCD contents. It is automatically updated when the simulator detects changes to the graphics memory.
The window also simulates the touch panel. Just place the mouse on the window, and press the left mouse button, to simulate a press or drag operation on the touch panel.
The background image size of the PV model can be set to fixed steps or to fill the window. This is set from the *Properties* dialog with the menu **Edit**, **Properties**.

Keys

The same keys as in the keyboard window can be used in this window.

7-12-9. Keyboard window

This window shows a keyboard layout for simulation of key presses.
To simulate a key press, just place the mouse on the wanted key, and press the left mouse button.
The background image size of the keyboard can be set to fixed steps or to fill the window. This is set from the Properties dialog with menu **Edit**, **Properties**.

Keys

The following keys can be used in this window.

↑	Simulate a press on the cursor up button on the PV unit keyboard
↓	Simulate a press on the cursor down button on the PV unit keyboard
←	Simulate a press on the cursor left button on the PV unit keyboard
→	Simulate a press on the cursor right button on the PV unit keyboard
Home	Simulate a press on the OK button on the PV unit keyboard

7-13. Help

7-13-1. Help menu

The Help menu displays information about the simulator.
The following entries are available in the Help Menu.

Help	Display help information for the window or dialog currently in use.
Contents	Display the help file contents.
About	Display information about the program version and copyright.

7-14. Additional menus

7-14-1. Popup menu

The Popup menu is available when right-clicking the mouse in a window.
The following entries are available in the Popup Menu, depending on the type of the window.

Run here	Run the program until the line where the cursor is located would be executed.
Set next instruction	Change the instruction pointer to another program line. This command is useful to repeat or skip some code lines due to a discovered bug in the program being debugged. Make sure the stack pointer and other registers are also updated if needed, when the program execution location is changed.
Goto	Display the *Goto* dialog box, to make it possible to go to another location in the current window.
Modify	Display the *Edit* dialog box, to change the value of the selected variable or memory location.
Show memory	Display the current location at the cursor in the *Source code*, *Global variables*, or *Local variables* window, as the corresponding address in the *Disassembled code* or *Memory* window.
Set breakpoint	Set a breakpoint on the line at the cursor.
Clear breakpoint	Clear the breakpoint on the line at the cursor.
Enable breakpoint	Enable the breakpoint on the line at the cursor. Enabled breakpoints are displayed as a filled circle.
Disable breakpoint	Disable the breakpoint on the line at the cursor. Disabled breakpoints are displayed as an unfilled circle.
Properties	Display the *Properties* dialog box, to see and change how the window contents are displayed.

8. Library Function For Application Program (Character Input & Drag Event)

8-1. Introduction

Each mode of Digital Diary consists of three states, "Data Input", "List Display", and "Data Display". This character-input library is used for "Data Input" and "Data Display".

8-2. About Functions

The character-input library is composed of the following function groups. As it is shown in the example below, three functions- "Initialization function", "Character string display function", and "Buffer operation function" - must always be used as a set. Data input and data display functions are provided on "Character string display function" and "Buffer operation function".

"LibTxtInit()" ·······························Initialization function (Common to data input and display)

This function initializes several variables for text input. This function must be called once when the display contents are loaded into the text buffer after the text area has been specified. (This function does not initialize the contents of the text buffer.)

"LibTxtInp()"·································Buffer operation function (For input.)

This is a routine, which is used to edit the contents in the text buffer. When the function receives a key code from the software keyboard process LibGetKeyM, it starts the text buffer write process.
Additionally, the function performs the drag selection process (cut, copy, paste) based on the internal touch waiting. The function also controls the scroll bar operation on the character-input screen.

"LibTxtDsp()"································ Character string display function (For input.)

This is a display routine during text input. This function displays characters, which have been input, and also switches the software keyboard automatically.

"LibTxtDspS()" ···························· Buffer operation function (For data display)

This function performs the drag (copy) process and scroll bar process for data display.

"LibTxtDspC()"···························· Character string display function (For data display)

This function is used to display data in each mode. The body is a core routine of the character display function during text input, LibTxtDsp( ).

8-3. About the Text Input Structure

In this library, several input conditions, such as pointer for the text buffer and display coordinates on the screen are specified in the structure for text input, TXTP.
The text-input structure contains work areas for the character-input process. Therefore, when specifying more than one structure, this makes it possible to have multiple independent text areas within the screen.

When a text display area is declared, a touch area for the character input library is also specified. Great care should be taken.
Actually, proportional characters are displayed in the specified area and character operations other than "Cursor move" and "Drag process (Copy and Paste)" provided by the character-input library become invalid in this area.

The following describes the contents of the structure for text input. All members used as input conditions for text input must be set before calling the "Initialization function (LibTxtInit())".
Initial setting of members defined as text input library work RAM is not needed since they are work areas for the character-input function.

```
/* General-purpose character input/data display parameter information */
typedef struct   TXTP {
        /* TextInputLibrary InputCondition */
        word         st_x;              /* TextArea Left X position */
        int          ed_x;              /* TextArea Right X position */
        word         st_y;              /* TextArea Up Y position     */
        int          it_y;              /* TextAreaLinePitch          */
        int          MAXGYO;            /* TextDisplayMaximumLineNumber */
        word         maxmj;             /* MaximumInputCharacterNumber  */
        byte         font;              /* FontSizeSpecification */
        bool         csen;              /* Cursor DisplayPermission */
        byte         rtnen;             /* LineFeedMark DisplayPermission */
        byte         *txbf;             /* TextBufferPointer      */
        word         *gdcmt;            /* GuidanceCommentsTablePointer */
        byte         *gdcmt2;           /* GuidanceCommentsBufferPointer */
        word         txtobj;            /* TextAreaTouchObject    */
        word         sbrobj;            /* ScrollBarTouchObject   */
        byte         *kwb_0;            /* KeywordBuffer0 Pointer */
        byte         *kwb_1;            /* KeywordBuffer1 Pointer */
        byte         *kwb_2;            /* KeywordBuffer2 Pointer */
        byte         *kw_dbk;           /* KeywordScreenShelterPointer */
        TCHTBL       *tchtb;            /* TouchTablePointer */
        T_SCR_POS    *s_pos;            /* ScrollBarPositionInformationPointer */
```

```
    /* TextInputLibraryWorkRAM */
    word        cp;             /* CursorPointer              */
    word        xp;             /* Display X position      */
    word        yp;             /* Display Y position      */
    word        c_xp;           /* Display Cursor X position */
    word        c_yp;           /* Display Cursor Y position */
    word        dlmp[22];       /* Line top character pointer */
    byte        ditp[22];       /* Line top character pointer */
    word        csln;           /* Cursor display line position */
    word        mjln;           /* Character draw start position */
    word        mjcnt;          /* Character Capacity in Text buffer */
    byte        ipdpst;         /* Input Display Status Flag */
    byte        txtst;          /* TextInput Status Flag */
    word        sr_sp;          /* Reverse start position pointer */
    word        sr_ep;          /* Reverse end position pointer */
    word        sr_xp;          /* Reverse end position pointer X */
    word        sr_yp;          /* Reverse end position pointer Y */
    word        srln;           /* Reverse line number                  */
    bool        selrv;          /* Select reverse flag     */
    word        ktyp;           /* For keyboard type shelter */
    byte        citm;           /* Now cursor item            */
    byte        wdcnt;          /* Input string length     */
    byte        kwnmt;          /* Candidate display flag */
    byte        kwsln;          /* Candidate display select line */
} TXTP;
```

*1········csen
FALSE(0x00): Cursor display disabled(Normally, this is used for data display)
TRUE (0x01): Cursor display enabled (Normally, this is used for data input.)
(0x02): Cursor display enabled && Keyword is registered, Display is allowed.
*2········rtnen
FALSE(0x00): Disables the CR code display. (Normally, this is used for data display)
RUE (0x01): Enables the CR code display. (Normally, this is used for data input.)
HALF (0x02): Disables the CR code input. (Normally, this is used for one line input.)
(0x03): Limits on the number of characters for each item.
(This is used for correction of CONTACTS and MEMO item names.)
(0x04): Hides the scroll bar display.
(This is used for full-screen display of MEMO.)
*3········For details of "gdcmt" and "gdcmt2", see the [About Guidance display function].
*4.·······They do not need to set if a value other than "0x02" is set in the member "csen" of the text input structure. See the chapter "10. About keyword registration" for more details.

8-4. About Text Buffer

The end code (0x00) must be put at the end of the text buffer. As a result, the buffer size becomes (the maximum number of characters + 1 byte).
Additionally, items are divided by the next item code (0xfe) if multiple items exist in one text buffer, like in the CONTACTS mode.

8-5. About Copy, Cut, and Paste

When making a character string highlighted to select it and copying, cutting, or pasting the selected character string, values shown below are set in the members of the structure "txtst" for text input before calling the input main function "LibTxtInp( )". (See the example for more details.)
Additionally, to copy a character string during data display, "txtst" is set to [TXTCOPY] before calling the function, "LibTxtDspS( )".

TXTCOPY:	Copies a highlighted character string selected to the copy buffer.
TXTCUT:	Deletes the selected character string from the text buffer after the copy process has been completed.
TXTPASTE:	Copies the character string in the copy buffer to the cursor position.

8-6. Basic Example (data input)

The example for performing a new input of data is shown below.
The specification prepared for this example is expected as follow: An icon that shows the transition to the data display is displayed on the screen, and the program steps out when the icon is touched.

```
/*** Designation of text area   ***/
#define   M_ST_X     10      /* st_x */
#define   M_ST_Y     13      /* st_y */
#define   M_ED_X     50      /* ed_x */
#define   M_IT_Y     9       /* it_y */
#define   M_MAXG     5       /* MAXGYO */

 /*** Scroll bar setting ***/
#define   SCR__Y             M_ST_Y               /* Scroll bar position   */
#define   SCR__SIZE          M_IT_Y*M_MAXG        /* Height of scroll bar   */
#define   SCR__DSP           M_MAXG             /* Number of display records */

byte      mtxbf[2049];               /* Buffer for text input   */
TCHTBLInptTch[3];                    /* Text touch area table   */
```

```
         /* Keyword saving buffer    */
        byte        keywd_0[33];
        byte        keywd_1[33];
        byte        keywd_2[33];
        byte        dspbk[400];

/*****************************************************************
        [Title]
        New data input (Keyword registration is provided.)
*****************************************************************/
void NewInp(void)
{
        byte kycd;
        TCHSTS          tsts;
        TXTP            m_in_tp;        /* Declaration of structure for text input*/
        T_SCR_POS       m_in_scb;       /* Scroll bar position information */

/*** Text input setting    ***/
        m_in_tp.st_x            = M_ST_X;       /* Start coordinate (X) of text display    */
        m_in_tp.st_y            = M_ST_Y;       /* Start coordinate (Y) of text display    */
        m_in_tp.ed_x            = M_ED_X;       /* End coordinate (X) of text display    */
        m_in_tp.it_y            = M_IT_Y;       /* Text display line spacing (Y) */
        m_in_tp.MAXGYO  = M_MAXG;       /* Number of text display lines    */
        m_in_tp.font            = IB_PFONT1;/* Display font type    */
        m_in_tp.csen            = 0x02;
                                /* Cursor display enabled(Keyword registration enabled) */
        m_in_tp.rtnen           = TRUE;         /* CR code display enabled    */
        m_in_tp.maxmj           = 2048;         /* Max number of allowable input characters */
        m_in_tp.txbf            = mtxbf;        /* Designation of text buffer address    */
        m_in_tp.gdcmt           = telgd;        /* Guidance comment table    */
        m_in_tp.txtobj          = OBJ_TXTTCH;   /* Object when text area is touched.    */
        m_in_tp.sbrobj          = OBJ_SCR_BAR; /* Object when the scroll bar is touched. */
        m_in_tptp.kwb_0 = keywd_0;      /* Keyword saving buffer 0 */
        m_in_tptp.kwb_1 = keywd_1;      /* Keyword saving buffer 1 */
        m_in_tptp.kwb_2 = keywd_2;      /* Keyword saving buffer 2 */
        m_in_tptp.kw_dbk= dspbk;        /* Keyword display saving area    */
        m_in_tp.tchtb           = InptTch;      /* Pointer for text scroll bar area    */
        m_in_tp.s_pos           = &m_in_scb;    /* Pointer for text and scroll bar inf. */

        mtxbf[0] = 0;                   /* Initialization of text buffer    */

        LibTxtInit(&m_in_tp);           /* Initialization of text input    */
```

```
    /*** Text touch area setting ***/
            LibTchStackClr();
            LibTchStackPush( NULL );
            LibTxtTchSet(&m_in_tp);                /* Text touch area PUSH */

            LibTchStackPush(TchDataKey);
                                    /* Icon touch table for transition of data display */
            LibTchStackPush(TchHardIcon);    /* Hard icon touch table    */

            LibClrDisp();
            LibIconPrint(&TicnDataKey);     /* Icon disp for transition of data display */

    /*** Data input loop    ***/
            while(mem_st==NEW_INP){
                    if(LibTxtDsp(&m_in_tp)==TRUE);      /* Display during text input */
                                    LibPutDisp();

                    kycd = LibGetKeyM(&tsts);                /* Waiting for software key    */

                    if(kycd==KEY_NONE){

/*** Process when a portion other than the text area, software keys,
     and scroll bar is touched. ***/
                            if(tsts.obj==OBJ_IC_DATAKEY){
                                            /* When touching "Data display" icon. */
                                    if(LibIconClick(&TicnDataKey,&tsts)==TRUE){
                                            mem_st = DATA_DSP;        /* To data display    */
                                                    LibTchInit();
                                    }

                            }else if(tsts.obj==OBJ_IC_CLRKEY){   /* When touch "CLR" icon. */
                                    if(LibIconClick(&TicnClrKey,&tsts)==TRUE){
                                            TelTextInit();         /* Initialization of text buffer    */
                                                    LibTxtInit(&tin_tp);
                                                            /* Initialization of text input    */
                                            }
                            }

    /*** Copy, cut, and paste process    ***/
                            }else if(tsts.obj == OBJ_HIC_CONT){
                                    mm = CpMenu();                        /* Copy & paste menu    */
                                    switch(mm){
                                            case 0:
                                                    m_in_tp.txtst = TXTCUT;  /* Cut    */
```

```
                                                  break;
                                          case 1:
                                          m_in_tp.txtst = TXTCOPY;/* Copy*/
                                                  break;
                                          case 2:
                                          m_in_tp.txtst = TXTPASTE;       /* Paste */
                                                  break;
                                  }
                          }
                  }

          /*** Main process of text input   ***/
                  LibTxtInp(kycd,&tsts,&m_in_tp);
          }
          LibTchStackClr();
}
```

8-7. Basic Examples (data display)

The following example is for the data display.

```
/*** Designation of text area   ***/
   #define   M_ST_X    10                          /* st_x */
   #define   M_ST_Y    13                          /* st_y */
   #define   M_ED_X    50                          /* ed_x */
   #define   M_IT_Y    9                           /* it_y */
   #define   M_MAXG    6                           /* MAXGYO */

   #define    SCR__Y          M_ST_Y          /* Scroll bar position */
   #define    SCR__SIZE       M_IT_Y*M_MAXG       /* Height of scroll bar */
   #define    SCR__DSP        M_MAXG          /* Number of display records */

   TCHTBL       DdspTch[3];                  /* Text touch area table   */

/************************************
          [Title]
                  Data display
************************************/
void DataDsp(void)
{
        TCHSTS       tsts;
        TXTP         m_dd_tp;        /* Declaration of structure for text input   */
        T_SCR_POS    m_dd_scb;       /* Scroll bar position information */
```

```
  /** Text input setting    **/
        m_dd_tp.st_x           = M_ST_X;    /* Start coordinate (X) of text display    */
        m_dd_tp.st_y           = M_ST_Y;    /* Start coordinate (Y) of text display    */
        m_dd_tp.ed_x           = M_ED_X;   /* End coordinate (X) of text display    */
        m_dd_tp.it_y           = M_IT_Y;             /* Text display line spacing (Y) */
        m_dd_tp.MAXGYO  = M_MAXG;            /* Number of text display lines    */
        m_dd_tp.font           = IB_PFONT1;        /* Display font type    */
        m_dd_tp.csen           = FALSE;               /* Cursor display disabled */
        m_dd_tp.rtnen          = FALSE;               /* CR code display disabled    */
        m_dd_tp.maxmj        = 100;        /* Max number of allowable input characters */
        m_dd_tp.txbf           = mtxbf;    /* Designation of text buffer address    */
        m_dd_tp.txtobj         = OBJ_TXTTCH; /* Object when text area is touched.    */
        m_dd_tp.sbrobj         = OBJ_SCR_BAR;
                                                       /* Object when the scroll bar is touched*/
        m_dd_tp.tchtb          = DdspTch; /* Pointer for text scroll bar area    */
        m_dd_tp.s_pos          = &m_dd_scb; /* Pointer for text and scroll bar inf. */

        LibTxtDspInit(&m_dd_tp);                     /* Initialization of text input    */

 /*** Text touch area setting ***/
        LibTchStackClr();
        LibTchStackPush( NULL );
        LibTxtTchSet(&m_dd_tp);                      /* Text touch area PUSH */

        LibTchStackPush(TchNewKey);
                    /* Icon touch table for transition of new input.    */
        LibTchStackPush(TchHardIcon);              /* Hard icon touch table    */

        LibClrDisp();
        LibIconPrint(&TicnNewKey);
                    /* Icon display for transition of new input.    */

        LibPutDisp();

/*** Data display loop    ***/
        while(mem_st==DATA_DSP){
            if(LibTxtDspC(&m_dd_tp)==TRUE)              /* Text display    */
                LibPutDisp();
            LibTchWait( &tsts );                               /* Waiting for touch    */
            switch(tsts.obj){
                case OBJ_IC_NEWKEY:            /* New input icon is touched */
                    if(LibIconClick(&TicnNewKey,&tsts)==TRUE){
                        mem_st = NEW_INP;
                                          /* Exit data display loop,and go to new input */
```

```
                                LibTchInit();
                        }
                        break;
                    case OBJ_HIC_CONT:
                        if(CpMenu()==1)                         /* Copy & paste menu    */
                            m_dd_tp.txtst = TXTCOPY;            /* Copy */
                        break;
                }
                /*** Text touch process during data display    ***/
                LibTxtDspS(&m_dd_tp,&tsts);
            }
            LibTchStackClr();
        }
```

8-8. About Guidance Display Function

The "Guidance display" specification that prompts user to input data is now available. However, note that this specification is valid when the text buffer is empty, or when there is a cursor but no text (no character) is available within the item separated by next item code (0xfe).

To use the guidance display function, transferring the pointer for the table containing the comment numbers used for the guidance display to the member, "word *gdcmt", of the text input structure will display the guidance automatically. The number of comment numbers in this table must be matched with that of items in the text buffer (normally, one item in modes other than the TEL mode). (Normally, only one word is used in modes other than the TEL mode.)
At this time, "word *gdcmt2" is ignored and therefore does not need to be specified.

(Examples of usage1)

```
     /*** TEL guidance table    ***/
    word telgd[] = {
            213,        /* NAME             */
            202,        /* ADDRESS_(H) */
            203,        /* FAX_(B)         */
            204,        /* PHONE_(B)      */
            205,        /* E-MAIL          */
            206,        /* EMPLOYER       */
            207,        /* FAX_(H)         */
            208,        /* PHONE_(H)      */
            209,        /* MOBILE          */
            210,        /* NOTE            */
    };
```

See the chapter "8.6. Basic examples (data input)" for more about this.

Furthermore, to make the guidance display inactive, "0xffff" is put in the table and the pointer for this table is transferred. To make the guidance display inactive, "0xffff" is put in the table and the pointer for this table is transferred. Additionally, the guidance display does not function during data display. Therefore, it is not necessary to particularly specify "word *gdcmt" during data display.

◆ The guidance display specification for the CONTACTS mode is supported.

To use desired character strings instead of the built-in (fixed) guidance message, the pointer for the table, in which "0xfffe" is put, is specified in "word *gdcmt". Additionally, the pointer for the top of the buffer where a desired character string is input to "word *gdcmt2" is specified.
In the buffer, messages for the number of items is divided by "0xfe" and the end code "0x00" must be put at the end of the messages.

8-9. About Text Area and Scroll Bar

It is necessary to set up the text area touch table and scroll bar touch table when using the text input function. At the same time, the object codes set to them must be registered to the members (txtobj, sbrobj) of the text-input structure.
This makes it possible to recognize which part of the text area (scroll bar) is touched in the inside of the text input function after it is called many times.

In order to prevent any mismatch between the touch table of the scroll bar area settled and the scroll bar actually displayed, now the function "LibTxtTchSet" that makes it possible to set up the touch table for the text area and the scroll bar area is available. Call this function once after calling the function "LibTxtInit" (text input initialization function).
Additionally, the pointer for the scroll bar position information used for text input (display) must be set in the member "*s_pos" of the text input structure.
Three empty structures (TCHTBL) for the touch table are prepared and the pointers for these structures are set in the member "tchtb" of the text-input structure.

8-10. About Keyword Registration

PocketViewer provides a function that registers and displays keywords when data is input in the scheduler mode.
To use this function, "0x02" is first set in "csen". At the same time, three buffers (33 bytes) for display of prospective keywords are prepared and pointers for these buffers are set in "kwb_0", "kwb_1", and "kwb_2".
Additionally, a buffer with a capacity of 400 bytes for temporary saving of the screen is prepared and its pointer is set in "kw_dbk".
If the keyword registration is not used ("csen" is either "0x00" or "0x01"), it is not necessary to set these pointers. See the chapter "8.6. Basic example (data input)" and "8.7. Basic examples (data display)" for the examples of usage.

<Remark>
There is a specification to perform a keyword registration by "SET" or "ESC" button during data input. To achieve this specification, call the function "LibTxtKeyWordSet(TXTP *tp)" when the button mentioned above is touched.

8-11. About Current Date/Time Paste Function

The function to insert the current date and time as a character string during data input is available.
To make this function active, the member "byte txtst" of the text input structure is set to "TXTDYTIM".
(This is the same as that "txtst" is set to "TXTPASTE" (txtst = TXTPASTE) when making the normal paste function active.)
The current time paste button is to be provided on the input screen of each application, and the application is programmed so that "txtst=TXTDYTIM" is set when such button is touched. (12/24-time format is supported.)

8-12. About dlmp/ditp

"dlmp[n]" and "ditp[n]" are provided in the work RAM which is used by the character input library. When referring to them after the key waiting process, it is possible to know which character in the text buffer is currently displayed on each line of the screen or which item is displayed.
This can be used to display the item name during text input or data display.

word dlmp[n]··················	Start character position of line "n" on the currently displayed screen. (For example, when 5th character is at the start position, this data is [0x0004].) If the buffer is empty, "0xffff" is set.
byte ditp[n]·····················	The number of items in "n" line on the currently displayed screen. (For example, when three items exist, this data is [0x02].) However, if an item extends over more than one line, this data is [0xff] in lines other than the start line. dlmp[0] shows the start of the line before the top line displayed on the screen (line currently not displayed).

(Example) TEL mode input screen

("↲" is for CR, and "■" is for a blinking cursor.)

```
0  0xffff      0xff
                      ================================================+
1  0x0000      0x00   |NAME         |S. SUZUKI                        |
                      +-------------+---------------------------------+
2  0x0009      0x01   |EMPLOYER     |CASIO                            |
                      +-------------+---------------------------------+
3  0x000f      0x02   |PHONE        |0123-45-6789                     |
                      +-------------+---------------------------------+
4  0x001c      0x03   |FAX          |0123-45-678                      |
                      +-------------+---------------------------------+
5  0x0029      0x04   |ADDRESS      |3-2-1 SAKAECHO↲                  |
                      +-------------+---------------------------------+
6  0x0037      0xff   |             |HAMURA-SHI↲                      |
                      +-------------+---------------------------------+
7  0x0042      0xff   |             |TOKYO, JAPAN                     |
                      +-------------+---------------------------------+
8  0x004d      0x05   |E-MAIL       |suzuk■                           |
                      +-------------+---------------------------------+
9  0xffff      0xff   |             |                                 |
                      ================================================+
10 0xffff      0xff
```

In this example, the contents of the text buffer are as follows:

```
53, 2E, 53, 55, 5A, 55, 4B, 49, FE,
43, 41, 53, 49, 4F, FE,
30, 31, 32, 33, 2D, 34, 35, 2D, 36, 37, 38, 39, FE,
30, 31, 32, 33, 2D, 34, 35, 2D, 36, 37, 38, 39, FE,
33, 2D, 32, 2D, 31, 20, 53, 41, 4B, 41, 45, 43, 48, 4F, 0D,
48, 41, 4D, 55, 52, 41, 2D, 53, 48, 49, 0D,
54, 4F, 4B, 59, 4F, 2C, 4A, 41, 50, 41, 4E, FE,
73, 75, 7A, 75, 6B, 00
```

8-13. Cautions for Use

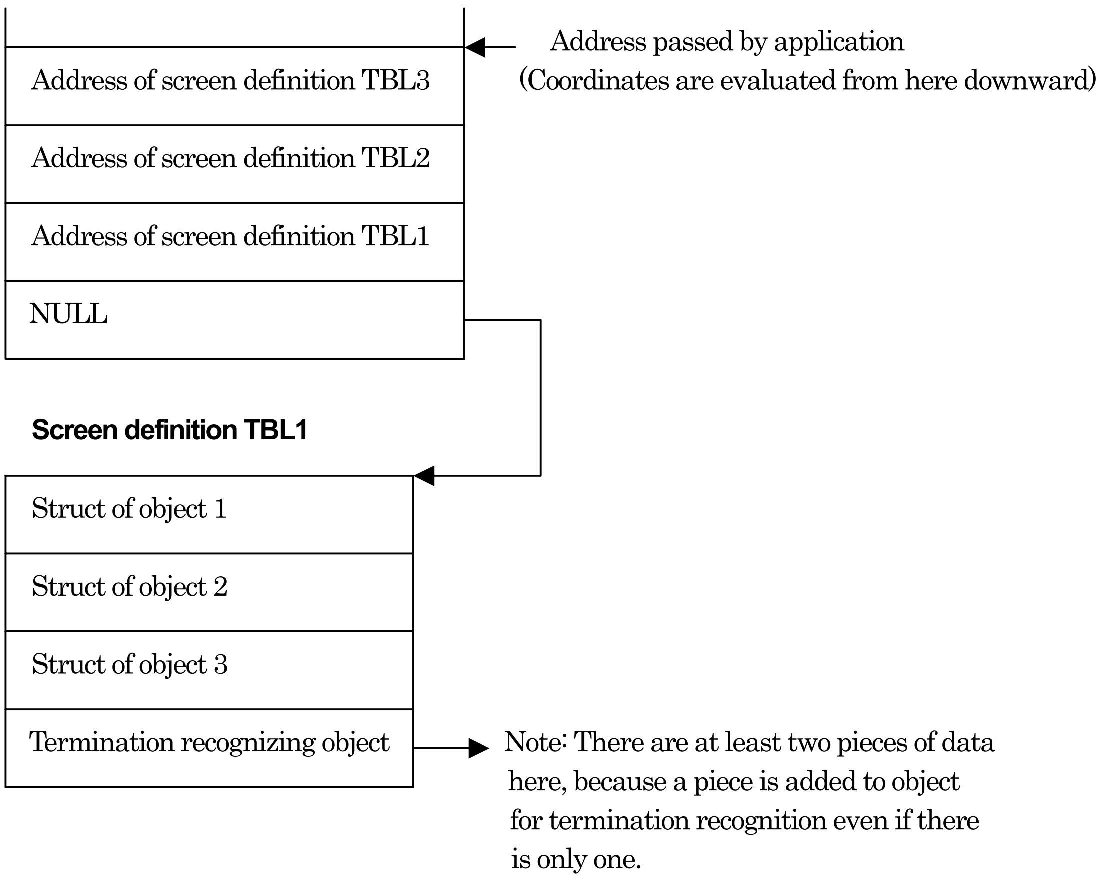
(1) To use the above function during data display or correction (copy) data input, the text initialization function "LibTxtInit (or LibTxtDspInit)" must be called after the text data is loaded into the text buffer. If the initialization function is called before loading data into the text buffer in the application program, the initialization function must be called again every time data has been loaded. (No bad impact is expected.)

(2) In the text initialization function, the number of characters in the text buffer is counted. However, if the initialization is made before loading data, the number of characters may become incorrect depending on the previous contents of the buffer. This may cause incorrect operation such as that dragging cannot be made completely. (Be careful because it looks as if the display is correctly done.)

(3) It is absolutely necessary to define the member "TCHTBL *tchtb /* Pointer for touch table */", one of input conditions for the text input structure. IF THIS IS NOT DEFINED, THE RAM WILL BE DESTROYED. THOUGH IN THE MOST TIME IT WORKS PROPERLY.

9. Library Function (Event)

9-1. Outline of event management by screen definition


Address of screen definition TBL3
Address of screen definition TBL2
Address of screen definition TBL1
NULL
Address passed by application
(Coordinates are evaluated from here downward)
Screen definition TBL1
Struct of object 1
Struct of object 2
Struct of object 3
Termination recognizing object
Note: There are at least two pieces of data here, because a piece is added to object for termination recognition even if there is only one.

```
Struct of object 1
typedef struct tchtbl{
        int             x1;     // Object scope X1
        int             y1;     // Object scope Y1
        int             x2;     // Object scope X2
        int             y2;     // Object scope Y2
        unsigned long   act;    // Action code
        unsigned short obj;     // Object code
        unsigned short ext;     // Termination code
                                  // (additional information)
}TCHTBL;
```

Figure 1: Structure of object definition table

- Memory arrangement for data of object definition (big endian)

x1	y1	x2	y2	act	obj	ext
12345678h	12345678h	12345678h	12345678h	12345678h	1234h	1234h

12,34,56,78　12,34,56,78　12,34,56,78　12,34,56,78　12,34,56,78　12,34　12,34

Small　A(C)　$\longleftarrow$ Address $\longrightarrow$　A(R)　Large

9-1-1. Description of object definition parameters

x1, y1, x2, y2: Coordinates of object scope (specify a rectangle)
Upper left coordinates (x1, y1) - Lower right coordinates (x2, y2)

- act: Action identification code

Code	Abbreviation	Description of action
IMUL_ACT_MK	touch MaKe	Moment when touched
IMUL_ACT_MV	touch MoVe	Move during touch
IMUL_ACT_MV_OUT	touch MoVe OUT	Moment when move out of object during touch
IMUL_ACT_MV_IN	touch MoVe IN	Moment when move into object during touch
IMUL_ACT_DW	touch DoWn	Time during touch
IMUL_ACT_DW_IN	touch DoWn IN	Time in object during touch
IMUL_ACT_BK	touch BreaK	Moment when touch is broken
IMUL_ACT_BK_IN	touch Break IN	Moment when touch is broken in object
IMUL_ACT_REP	touch REPeat	Pass repeat interval during touch
IMUL_ACT_500M	500Msec	500 msec update occurs for clock time
IMUL_ACT_ALM	AlarM	Alarm coincidence occurs
IMUL_ACT_INPUT	INPUT	Input processing
IMUL_ACT_NONE	NONE	Specify invalid area
IMUL_ACT_TMR	TiMeR	Timer coincides
IMUL_ACT_1SEC	1SEC	Update 1 second for clock
IMUL_ACT_APO	APO	APO
IMUL_ACT_CMD	CMD	Application control command
IMUL_ACT_SYSWINCLS	SYStemWindowCLoSe	Terminate display of alarm/alert

- obj: Object code

0000h	NULL (2 bytes) for object as terminator (stopper) of screen definition table
0001h ~ 7FFFh	Reserved for system/common
8000h ~ FFFFh	Freely set by application

- ext: Code for extension

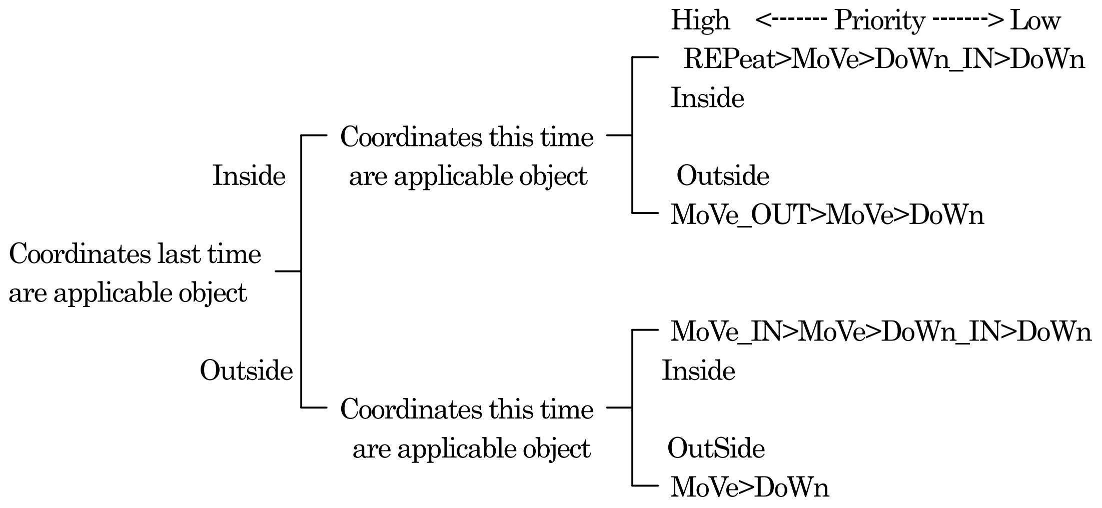
0000h	Reserved code for extension (usually NULL)

9-1-2. Output of generating object

Contents of touch information struct

```
typedef struct TCHSTS {
    unsigned short obj;        // Object code
    unsigned long act;         // Action code
    int x;                     // Touch position coordinate (vertical)
    int y;                     // Touch position coordinate (longitudinal)
    unsigned short ext;        // Additional information that occurs (usually NULL)
    unsigned char istr[4];     // <Unused>
} TCHSTS;
```

Idea of priority for action identification (while acquiring coordinates after MAKE)


High <------- Priority -------> Low
REPeat>MoVe>DoWn_IN>DoWn
Inside
Coordinates this time are applicable object
Inside
Outside
MoVe_OUT>MoVe>DoWn
Coordinates last time are applicable object
MoVe_IN>MoVe>DoWn_IN>DoWn
Inside
Outside
Coordinates this time are applicable object
OutSide
MoVe>DoWn

[Reference 1] BIOS output of generating object

BIOS output is shown below. This is used inside libraries.

Contents of struct for output

```
typedef struct eventsts {
    unsigned short obj;         // Object in which event occurs
    unsigned long act;          // Any of actions of generating object
    union {
        PEN_LCD pen;            // Touch information
        KEY_DAT key;            // Key information
        TIMESYS time;           // Alarm time information (alarm BIOS definition)
       APL_CMD command;         // Application control command
    } data;
    unsigned short ext;         // Additional information that occurs (usually NULL)
    unsigned char istr[4];      // For extension (usually NULL)
    unsigned char pwon;         // Power on information
} EVENTSTS;
```

Touch information struct

```
typedef struct pen_lcd {
   int lcd_x;               // X coordinate (LCD coordinate)
   int lcd_y;               // Y coordinate (LCD coordinate)
   AD_VALUE ad;                 // A/D value
} PEN_LCD;
```

Key information struct

```
typedef struct key_dat {
    unsigned short kcode;    // Key code (command code)
    unsigned short ccode;    // Character code
} KEY_DAT;
```

9-2. Classification of object identification code

Code		
0x0000	Terminal record specification	Object code defined in system Application cannot set these codes
0x0001 ⋮ 0x7FFF	Reserved for system/common	
0x8000 ⋮ 0xFFFF	Freely set by application	Code range that can be freely set by applications

9-2-1. Terminal record specification (0x0000)

Object for judging termination that is set at the end of an object definition array as the terminator of a screen definition table.

[Object code]	[Name]	Remark
OBJ_END	TBL terminator	Termination code of screen definition table

9-2-2. Reserved for system/common item (0x0001A~0x7FFF)

An object dependent on hardware such as a hardware icon or an object used in the BIOS and other things that are provided from the system to applications belongs to and is defined in any of the following classes:

- Object in hardware icon area (OFF, menu, etc.)
- Object of lever push switch
- Object of 500 msec output
- Object of alarm coincidence
- Special object such as break key sample or touch scan

Hardware icon See <Auxiliary 1>

Fixed object that is already printed on the panel literally like hardware.

Object code	Name	Remarks
OBJ_HDICON1	Hardware icon 1	Position 1
OBJ_HDICON2	Hardware icon 2	Position 2
OBJ_HDICON3	Hardware icon 3	Position 3
OBJ_HDICON4	Hardware icon 4	Position 4
OBJ_HDICON5	Hardware icon 5	Position 5
OBJ_HDICON6	Hardware icon 6	Position 6
OBJ_HDICON7	Hardware icon 7	Position 7
OBJ_HDICON8	Hardware icon 8	Position 8
OBJ_HDICON9	Hardware icon 9	Position 9
OBJ_HDICON10	Hardware icon 10	Position 10
OBJ_HDICON11	Hardware icon 11	Position 11
OBJ_HDICON12	Hardware icon 12	Position 12
OBJ_HDICON13	Hardware icon 13	Position 13
OBJ_HDICON14	Hardware icon 14	Position 14
OBJ_HDICON15	Hardware icon 15	Position 15

Action key See <Auxiliary 2>

Object code	Name	Remarks
OBJ_LPSW_PUSH	PUSH	
OBJ_LPSW_UP	UP	
OBJ_LPSW_DOWN	DOWN	
OBJ_LPSW_LEFT	LEFT	
OBJ_LPSW_RIGHT	RIGHT	

Key

Object code	Name	Remarks
OBJ_ALL_KEY	All keys	
OBJ_CODE_KEY	Code key	Key code: 00h-7Fh
OBJ_CTRL_KEY	Control key	Key code: 80h-FFh

500 msec output (independent occurrence) See <Auxiliary 3>

Object code	Name	Remarks
OBJ_SYS_500MSEC	500 msec event	Output depending on CPU clock count

Alarm coincidence output See <Auxiliary 4>

Object code	Name	Remarks
OBJ_ALM_ALARM	Alarm coincidence	
OBJ_ALM_CLOSE	Request for closing	Alarm coincidence during alarm indication　alarm indication
OBJ_ALM_QUIT	Request for emergency closing　of alarm indication	Emergency operation actuated during alarm indication

BLD detection See <Auxiliary 5>

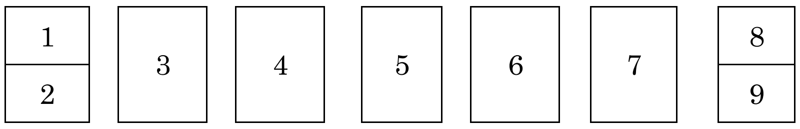
Object code	Name	Remarks
OBJ_SYS_BLD1	BLD1 detected	
OBJ_SYS_BLD1CLOSE	Request for closing BLD1 indication	

Closing system window See <Auxiliary 6>

If there is an applicable object when alarm or alert is indicated, the applicable object and system window closing action are output.

<Auxiliary 1> Hardware icon

Positions of hardware icons are as follows:



Define coordinates of a hardware icon object as fixed to (0,0,0,0).
An example of object definition table is shown below:

```
static TCHTBL TchHardIcon[] = {
    0,0,0,0, IMUL_ACT_MK, OBJ_HDICON1, 0x0000, /* Position 1 */
    0,0,0,0, IMUL_ACT_MK, OBJ_HDICON2, 0x0000, /* Position 2 */
    0,0,0,0, IMUL_ACT_MK, OBJ_HDICON3, 0x0000, /* Position 3 */
    0,0,0,0, IMUL_ACT_MK, OBJ_HDICON4, 0x0000, /* Position 4 */
    0,0,0,0, IMUL_ACT_MK, OBJ_HDICON5, 0x0000, /* Position 5 */
    0,0,0,0, IMUL_ACT_MK, OBJ_HDICON6, 0x0000, /* Position 6 */
    0,0,0,0, IMUL_ACT_MK, OBJ_HDICON7, 0x0000, /* Position 7 */
    0,0,0,0, IMUL_ACT_MK, OBJ_HDICON8, 0x0000, /* Position 8 */
    0,0,0,0, IMUL_ACT_MK, OBJ_HDICON9, 0x0000, /* Position 9 */
    0,0,0,0, IMUL_ACT_NONE, OBJ_END,     0x0000   /* End identification */
};
```

<Auxiliary 2>　Action key

<Input>

An action key is enabled by defining an object in the touch stack like a hardware icon or other events. The object definition table is as follows:

```
static TCHTBL ActionKey[] = {
    0,0,0,0, IMUL_ACT_MK                   , OBJ_LPSW_PUSH , 0x0000,
    0,0,0,0, IMUL_ACT_MK | IMUL_ACT_REP, OBJ_LPSW_UP     , 0x0000,
    0,0,0,0, IMUL_ACT_MK | IMUL_ACT_REP, OBJ_LPSW_DOWN , 0x0000,
    0,0,0,0, IMUL_ACT_MK | IMUL_ACT_REP, OBJ_LPSW_LEFT , 0x0000,
    0,0,0,0, IMUL_ACT_MK | IMUL_ACT_REP, OBJ_LPSW_RIGHT, 0x0000,
    0,0,0,0, IMUL_ACT_NONE, OBJ_END,    0x0000
};
```

<Output>

Output contents of struct when an action key is pressed (TCHSTS)

.obj	Any of OBJ_LPSW_PUSH, OBJ_LPSW_UP, OBJ_LPSW_DOWN, OBJ_LPSW_LEFT or OBJ_LPSW_RIGHT
.act	IMUL_ACT_MK or IMUL_ACT_REP
.x	Undefined
.y	Undefined
.ext	Additional information that occurs (usually NULL)
.istr[0]-[4]	For extension (usually NULL)

[Note]
You don't need to input coordinate values and leave them fixed to NULL. Define the above five objects as one set. You don't need to define power on by lever push in the touch stack. When it occurs, object OBJ_LPSW_PUSH is unconditionally output.

<Auxiliary 3>　500 msec event

<Input>

The object definition table is as follows:

```
static TCHTBL 500msec[] = {
      0,0,0,0, IMUL_ACT_500M, OBJ_SYS_500MSEC, 0x0000,
      0,0,0,0, IMUL_ACT_NONE, OBJ_END,    0x0000
};
```

<Output>

There are cases in which a 500 msec event occurs independently and cases in which it occurs during another object event. Outputs in each case are shown below. (TCHSTS)

(1) When 500 msec event occurs independently

.obj	OBJ_SYS_500MSEC
.act	IMUL_ACT_500M
.x	Undefined
.y	Undefined
.ext	IM_EXT_500M
.istr[0]-[4]	NULL

(2)　When 500 msec event occurs simultaneously with another event

.obj	Applicable object code
.act	IMUL_ACT_500M
.x	Undefined
.y	Undefined
.ext	IM_EXT_500M
.istr[0]-[4]	NULL

[Note]

By adding the object definition shown in <Input> above to the screen definition table used when calling a usual event BIOS, a 500 msec event output is considered to be specified by an application, and items shown in <Output> above are output when an event occurs.

[Restrictions]

Because the 500 msec count is dependent on the hardware timer (CPU clock), it is output in half a cycle that the clock updates the second.

<Auxiliary 4> Alarm coincidence event

<Input>
It is prerequisite that the alarm has been set by the alarm setting BIOS.

<Output>
Outputs when an alarm coincidence occurs are as follows. (TCHSTS)

.obj	OBJ_ALM_ALARM
.act	IMUL_ACT_ALM
.x	Undefined
.y	Undefined
.ext	IM_EXT_NULL
.istr[0]-[4]	NULL

[Note]
An alarm coincidence is output only to the alarm task, not to usual applications.
However, when the alarm window closes, a system window closing event is output to applications.

<Auxiliary 5> BLD indication

<Input>
None

<Output>
Outputs when a BLD detection related event occurs are as follows. (TCHSTS)

.obj	OBJ_SYS_BLD1 or OBJ_SYS_BLD1CLOSE
.act	IMUL_ACT_NONE
.x	Undefined
.y	Undefined
.ext	IM_EXT_NULL
.istr[0]-[4]	NULL

[Note]
Every BLD detection related event is output to the alert task, not to usual applications. OBJ_SYS_BLD1 is output when a BLD condition is detected. OBJ_SYS_BLD1CLOSE is output when a BLD indication closing event occurs. However, when the BLD window closes, a system window closing event is output to applications.

<Auxiliary 6>　Closing system window

<Input>
None in particular

<Output>
(TCHSTS)

.obj	Object that is current target of operation
.act	IMUL_ACT_SYSWINCLS
.x	Undefined
.y	Undefined
.ext	IM_EXT_NULL
.istr[0]-[4]	NULL

[Note]
Closing of the system window is notified to applications if there is an object that is the current target of operation when the alarm/alert window closes.
The output object is output as the object that is the current target of operation, and output action is output as IMUL_ACT_SYSWINCLS.

9.3. Break key sample

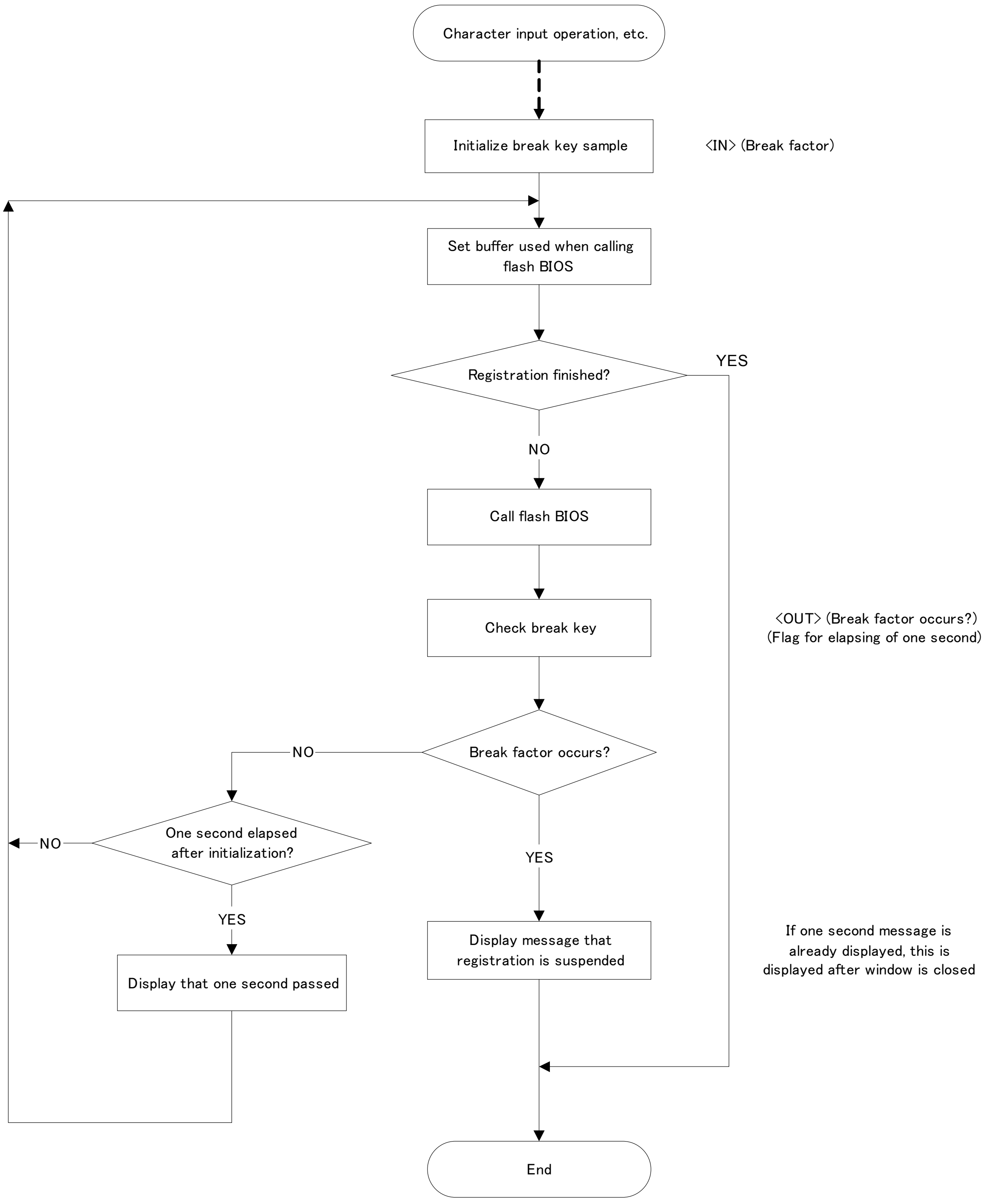
Break factor
byte b_smp; Valid break factor. Setting one of the following factor bits enables break by each factor.

IX_BLD1MSG	BLD message level
IX_CRADLE	Cradle key
IX_ESCBRK	ESC touch

ESC icon touch coordinate struct for break key sample
Other than hardware icons, an ESC icon that enables break can be set. It is defined in the following struct.
ESC touch must be added as a break factor.

```
typedef struct BK_SMPL_TCH{
      int x1;    // Upper left coordinate (lateral)
      int y1;    // Upper left coordinate (longitudinal)
      int x2;    // Lower right coordinate (lateral)
      int y2;    // Lower right coordinate (longitudinal)
} BK_SMPL_TCH;
```

- Example for batch registration (communication), batch delete or search



Character input operation, etc.
Initialize break key sample
<IN> (Break factor)
Set buffer used when calling flash BIOS
Registration finished?
YES
NO
Call flash BIOS
Check break key
<OUT> (Break factor occurs?)
(Flag for elapsing of one second)
Break factor occurs?
NO
One second elapsed after initialization?
NO
YES
Display that one second passed
YES
Display message that registration is suspended
If one second message is already displayed, this is displayed after window is closed
End

Appendix A: Message Table

model:PV-S1600

No	Message(english)	No	Message(english)	No	Message(english)
0	Reminder	30	PC sync	60	Set
1	Schedule	31	All	61	Set date
2	To Do	32	Full screen	62	Set time
3	Contacts	33	Sound	63	Auto power off
4	Memo	34	Capacity	64	Week format
5	Expense	35	Memory management	65	Keyboard
6	Conversion	36	Language	66	Initial
7	Financial Cal	37	Contrast	67	Initial
8	Quick-Memo	38	Set date/time	68	Communication@setup error!
9	Game	39	Format	69	English
10	Pop Up Tools	40	Touch Panel Alignment	70	Español
11	World Clock	41	Data communication	71	Deutsch
12	Quick Schedule	42	Communication setup	72	Français
13	Set city	43	Yes	73	Italiano
14	New	44	No	74	Free
15	Edit	45	Power setting	75	Free
16	Search	46	Screen tap power on	76	Password edit
17	View	47	Screen Options	77	Password
18	System	48	Secret	78	Schedule alarm
19	Option	49	List	79	Reminder alarm
20	Cut	50	Use start-up screen	80	To Do alarm
21	Copy	51	Backlight time	81	Daily alarm
22	Paste	52	Rate table	82	Key tone
23	Delete	53	Rate edit	83	Cut
24	Item edit	54	Name edit	84	Up
25	Item copy	55	Name edit	85	Lighter
26	To secret area	56	All data items	86	Darker
27	To open area	57	Single data item	87	Send
28	Action Menu	58	Category data items	88	Receive
29	Spreadsheet	59	sec	89	Tools

No	Message(english)	No	Message(english)	No	Message(english)
90	Comp	120	Type	150	End Date
91	Saving	121	Weekly	151	
92	Loan	122		152	
93	CSM	123	1-month	153	
94	Euro	124	2-month	154	Category
95	Odd	125	3-month	155	Account Name
96	Business ←→ Personal	126	Alpha	156	Account Type
97	General	127	Num	157	Balance
98	Rate	128	To Do list	158	
99	Rounding	129	Reminder list	159	Daily
100	Currency	130		160	Day
101		131		161	Monthly 1
102	Model	132		162	Monthly 2
103		133	Multi-date item	163	Yearly 1
104	CODE	134		164	Yearly 2
105	SMBL	135	Highlight	165	
106	NEXT	136	Unhighlight	166	
107	CAPS	137		167	D
108	SPACE	138	Appointment	168	W
109	Esc	139	Specified data items	169	M1
110	Clr	140		170	M2
111	Exe	141		171	Y1
112	Serial	142		172	Y2
113	USB	143	Schedule data items	173	
114		144	Done data items	174	
115	Date	145	To Do data items	175	
116	Time	146	Reminder data items	176	
117		147	All Scheduler data items	177	From
118	Month	148	Description	178	To
119	Week	149	Start Date	179	SU

No	Message(english)	No	Message(english)	No	Message(english)
180	MO	210	Note	240	Card
181	TU	211	Position	241	Bank
182	WE	212	Department	242	
183	TH	213	Name	243	Prompt edit
184	FR	214	Free 1	244	
185	SA	215	Free 2	245	
186	S	216	Free 3	246	Untitled 1
187	M	217	Free 4	247	Untitled 2
188	T	218	Free 5	248	Untitled 3
189	W	219	Free 6	249	Untitled 4
190	T	220	Free 7	250	Untitled 5
191	F	221	Free 8	251	
192	S	222	Free 9	252	Font
193		223	Free 10	253	Memo?
194		224	Free 11	254	
195	Business	225	Free 12	255	
196	Personal	226	Free 13	256	
197		227	Account	257	Move
198		228	Transaction	258	Amount
199		229	All Accounts	259	Payment type
200		230	Checking	260	Expense type
201	Address (B)	231	Savings	261	Select type
202	Address (H)	232	Credit Card	262	Label edit
203	Fax (B)	233	Cash	263	Total
204	Phone (B)	234	Total Balance	264	Balance(*)
205	E-mail	235	Info.	265	Not Cleared
206	Company	236	Account Infomation	266	Cleared
207	Fax (H)	237	Latest Balance	267	Reconciled
208	Phone (H)	238	Number	268	
209	Mobile	239	Payee	269	

No	Message(english)	No	Message(english)	No	Message(english)
270		290	October	310	Fri
271	Menu edit	291	November	311	Sat
272		292	December	312	Level1
273	Transfer file	293	Jan	313	Level2
274	OK to delete@the selected data?	294	Feb	314	Level3
275		295	Mar	315	Level
276	OS Update	296	Apr	316	High score
277	Operating system@will be updated.@Continue?	297	May	317	Congratulations!
278	Alarm	298	Jun	318	Batteries are getting@weak!@Replace them as@instructed in the@User's Guide.
279		299	Jul	319	
280		300	Aug	320	Data error!@Do you want to view@data before resetting?
281	January	301	Sep	321	
282	February	302	Oct	322	Data error!@Consult your User's@Guide for correct@procedure.
283	March	303	Nov	323	Data item not@found!
284	April	304	Dec	324	Memory is full!
285	May	305	Sun	325	
286	June	306	Mon	326	Input all@required data!
287	July	307	Tue	327	Make sure you@are inputting@the date@correctly.
288	August	308	Wed	328	Check your home time@setting and correct it@if necessary!
289	September	309	Thu	329	Deleting!

No	Message(english)	No	Message(english)	No	Message(english)
330	OK to delete all data@items?	340	The unit is being@initialized.@Please wait.	350	Transfer the data to@the secret memory@area?
331	OK to delete the data@item?	341	Data stored!	351	This procedure@registers a password@and creates a secret@memory area.
332	OK to delete all data@items in this category?@-category name-	342	Setting has been@made!	352	Register the@password?
333	Complete!@This image will@appear whenever@you power up.	343		353	Performing@memory@management!
334		344		354	This operation may@take a long time to@perform.@Do you want to@continue?
335		345	Wrong password!	355	Use the stylus to@touch the center of@each of the crosses@on the screen.
336		346	Check the secret@memory area!	356	Touch panel@alignment@complete!
337	Searching!	347	Password matches!	357	
338	The reset operation@deletes all data in@memory!@Do you want to@continue?	348	Password@registered!	358	
339	The next step starts@data deletion and unit@reset!@Do you want to@continue?	349	Transfer the data to@the open memory@area?	359	

No	Message(english)	No	Message(english)	No	Message(english)
360		390	AZERTY	420	Payment type?
361	1000 alarms are@already set!	391	QWERTZ	421	Expense type?
362	All dates you select@must be within the@same year.	392		422	Calendar
363		393	Week 1	423	
364	Unhighlight all dates in@the specified period?	394	Week 2	424	
365		395	Week 3	425	
366	That alarm time@is already passed!	396	Week 4	426	Sunday
367	That alarm time@is already used!	397	Week 5	427	Monday
368	Make sure you@are inputting the@time correctly.	398	Score	428	Tuesday
369	That date is@outside of the@specified period.	399	Scheduler	429	Wednesday
370		400	Period total	430	Thursday
371	You cannot specify@more than 60 repeats!	401	Version	431	Friday
372		402	Clock	432	Saturday
373	OK to delete all@Scheduler data items?	403	Menu	433	min
374	OK to delete all To Do@data items?	404	System tools	434	M/D/Y
375	OK to delete all done@data items?	405	Add in	435	D/M/Y
376	OK to delete all@Reminder data items?	406	System Library	436	Y/M/D
377	OK to delete all@Schedule data items?	407	PVOS for PV-S1600	437	Set up parameters
378	OK to delete all data@items within the@specified period?	408	Common Library	438	Parity
379	Move the memo to@the location below the@highlighted memo name?	409	Message & Font	439	Even
380		410	Graphic	440	
381	Please wait...	411	Swap	441	None
382	Ready to receive!	412		442	Bit length
383	Ready to send!	413	Date?	443	BPS
384	Receiving!	414	Time?	444	New High Score
385	Sending!	415	Alarm Time?	445	Game-1
386	Communication@error!	416	Description?	446	Game-2
387	Stopped!	417	Due Date?	447	OK
388	Data communication@in progress!	418	Alarm Date?	448	
389	QWERTY	419		449	Scheduler alarms

No	Message(english)	No	Message(english)	No	Message(english)
450	Latest calls	480	Payment		
451	Memory maintenance@can increase memory@for new storage.				
452	Change Home Time city?@Doing so can affect@Schedule data and@alarms.				
453	That name is@already used!				
454	Start Date?				
455	End Date?				
456	7				
457	8				
458	4800				
459	9600				
460					
461					
462	Pocket Viewer				
463	BUSINESS NAVIGATOR				
464	SF/CSF/NX				
465	PV Model				
466	BN Model				
467	SF/CSF/NX Model				
468	- Send -				
469	- Receive -				
470	Contacts (B)				
471	Contacts (P)				
472	Calculator				
473	(Other)				
474	Contacts (1~5)				
475	Memo (1~5)				
476	This operation clears@the latest calls list.@Do you want to@continue?				
477	Save				
478	Screen copy				
479	Start-Up Screen				

Appendix B: Character Code Table

CharacterCodeTable

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	NULL	¨	SPC	0	@	P	‘	p	Á	á	Ä	ä	ª	→	ı		0
1		´	!	1	A	Q	a	q	É	é	Ë	ë	º	√	þ		1
2		`	"	2	B	R	b	r	Í	í	Ï	ï	×		Þ		2
3		^	#	3	C	S	c	s	Ó	ó	Ö	ö	÷	§	ý		3
4		~	$	4	D	T	d	t	Ú	ú	Ü	ü	±	↑	Ý		4
5			%	5	E	U	e	u	À	à	Ã	ã	°	↕	•		5
6			&	6	F	V	f	v	È	è	Õ	õ	²	↓	¹		6
7			'	7	G	W	g	w	Ì	ì	Ñ	ñ	³	¯	€		7
8		◆	(	8	H	X	h	x	Ò	ò	Ĳ	ĳ	µ	ÿ	SYS	SYS	8
9	TAB	|	)	9	I	Y	i	y	Ù	ù	Æ	æ	½	®	SYS	SYS	9
A		[	*	:	J	Z	j	z	Â	â	Ç	ç	¼	«	SYS	SYS	A
B		]	+	;	K	[	k	{	Ê	ê	Å	å	¾	»	SYS	SYS	B
C		∫	,	<	L	\	l	¦	Î	î	Φ	φ	ƒ	©	SYS	SYS	C
D	↵	●	-	=	M	]	m	}	Ô	ô	ß	£	|	¤	SYS	SYS	D
E		🔔	.	>	N	^	n	~	Û	û	¶	¥	Fr	ð	SYS	SYS	E
F		◢	/	?	O	_	o		¡	¿	¢	Ω	←	Đ	SYS	SYS	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

10H - 1FH=DISP ONLY E8H - EFH , F8H - FFH=SYSTEM RESERVE

Appendix C: Library Function List

SHPV Library Function

For Application Program

Listed By Purpose

Mark	Detail
New!	**Library added for new PV.**
@	**Function that is left to secure compatibility with previous PV. There are some functions that are in NOP. Do not use these for new PV in principle.**
*	**be sure to INCLUDE following two files(for OLD PV compatibility).** com_lib\h\l_define.h,com_lib\h\l_libc.h
#	**Public library header is not included.**

■ Display Functions

Function name	Function
LibLine()	Draws a border.
LibMeshLine()	Draws a dotted line.
LibLineClr()	Clears a border.
LibLineCplmnt()	Draws a border by bit inversion.
LibDotOn()	Draws a dot.
LibDotOff()	Clears a dot.
LibPutProFont()	Displays a proportional font.
LibGetProFontSize()	Gets a size of proportional font data.
LibPutProStr()	Displays a proportional font string.
LibGetProStrSize()	Gets a size of proportional font string.
LibPut35Font()	Displays a 3 * 5 font.
LibPut35Str()	Displays a 3 * 5 font character string.
LibReverse()	Reverses a rectangular area display.
LibMesh()	Shades a rectangular area display.
LibBox()	Draws a box.
*LibSKeyRevSub()	Provides various appearances when pressing a software key.
LibGdsBox()	Draws a box by overriding.
LibGdsBoxMesh()	Draws a box with dotted line.
LibGdsBoxClr()	Clears a box.
LibGdsBoxCmp()	Draws a box using XOR operator.
LibGdsClr()	Clears a rectangular area.
LibGdsReverse()	Reverses a box area.
LibGdsMesh()	Shades a box area.
LibGdsDotOn()	Draws a dot.
LibGdsDotOff()	Clears a dot.
LibGdsDotCmp()	Draws a dot using XOR operator.
LibGdsLine()	Draws a line by overriding.
LibGdsLineClr()	Clears a line.
LibGdsLineMesh()	Draws a dotted line.
LibGdsLineCmp()	Draws a line using XOR operator.
*LibPutBoxSub()	Draws a box with a type specified.
*LibCngeBoxSub()	Change the appearance of the rectangular area with a type specified.
*LibPutDotSub()	Draws a dot with a type specified.
*LibPutLineSub()	Draws a line with a type specified.
LibPutGraph()	Displays a graphic data.
LibPutGraphM()	Displays a graphic data with a shading.
LibPutGraphO()	Displays a graphic data with the write-mode specification.
LibPutFarData()	Displays built-in graphic data.
LibGetGraph()	Gets the VRAM contents for a rectangular area.
LibGrpUp()	Scrolls up a rectangular area.
LibGrpDwn()	Scrolls down a rectangular area.
LibPutDisp()	Transfers VRAM data to D/D.(Entire screen)
LibPutDispBox()	Transfers VRAM data to D/D.(Area specification)
LibClrDisp()	Clears the screen contents. (Entire screen)
LibClrBox()	Clears the screen contents. (Area specification)
LibSetDispMode()	Sets the write mode to D/D.
LibInitDisp()	Initializes a screen.(Text)
LibGrphUpSideDown()	Flips a graphic data vertically.
LibStringDsp()	Displays a proportional character string for listing.

■ Window Functions

Function name	Function
LibOpenWindow()	Opens a window.
LibOpenWindowS()	Opens a window.(Coordinates can be changed. With border options.)
LibCloseWindow()	Closes the window opened last.

■ Touch Functions

Function name	Function
LibTchInit()	Initializes the touch information.
LibTchStackClr()	Clears the stack contents of the touch information table.
LibTchStackPop()	Gets the touch information table registered last.
LibTchStackPush()	Registers the touch information table.
*LibTchHardIcon()	Performs the process for the hardware icons.
LibTchWait()	Gets the touch information table. (Waits for touching)
New!LibTchWaitScan()	Scan touch state
LibIconPrint()	Displays an icon.
LibIconPrintR()	Performs the reverse display of the icon.
LibIconPrintM()	Displays an icon with shading.
LibIconClick()	Controls touching of the icon information.
LibIconClick2()	Controls touching of the icon information. (For no shadow pattern.)
LibScrollPrint()	Displays the scroll bar.
LibScrollArrowPrint()	Displays the up and down arrows on the scroll bar.
LibScrollClick()	Controls touching of the scroll bar.
LibScrPosCheck()	Checks a scroll bar touch position.
LibKeyInit()	Initializes the generic keyboard.
LibDispKey()	Displays the generic keyboard.
LibGetKeyM()	Performs the generic keyboard waiting.
LibGetCale()	Performs the calendar software keyboard process.
LibInputTime()	Performs the time software keyboard process.
LibInputTimeBar()	Performs the time bar term input process.
LibInputTerm()	Performs the term input process.
*LibSKeyRev()	Provides the pressed appearance only to the software keyboard.
*LibSKeyIsCd()	Checks a object code in the software keyboard.
LibIconMoveDown()	Provides the pressed appearance to the icon.
LibIconMoveUp()	Provides the up-transition appearance to the icon.
LibBkSampleInit()	Initializes the break-key sample.(For the main unit process)
LibBkSampleCheck()	Monitors the break-key sample status.
*LibBkSampleInitSub()	Initializes the break-key sample.(Body)
LibBlockIconClick()	Performs the block-type icon click process.
LibRepOff()	Turns off the touch repeat.

■ FLASH Functions

Function name	Function
LibFileFindNext()	Searches the FLASH memory for next data.
LibFileFindPrev()	Searches the FLASH memory for previous data.
LibFileFindNextExt()	Searches the FLASH memory for next data.(Extended version.)
LibNextSearchCld()	Searches the FLASH memory for next data.(Only for Calendar)
LibFileRead()	Reads data from the FLASH memory
LibFileWrite()	Writes data to the FLASH memory.
LibFileCorect()	Corrects the FLASH data without changing the FLASH pointer.
LibFileRemove()	Deletes data from the FLASH memory.(1 record)
LibFileRemoveAll()	Deletes data from the FLASH memory.(All records in specified modes)
LibGetFileInfo()	Gets the FLASH memory data information.
LibGetFileCnt()	Gets the number of records in the specified mode registered in the FLASH memory.
LibGetFlash()	Gets the total capacity of the FLASH memory.
LibGetFreeBlock()	Gets the number of free blocks of the FLASH memory.
LibGetDataCond()	Checks the FLASH data status.
LibFileRemake()	Executes the FLASH memory remaking process.
LibFileExch()	Sorts FLASH data. (Moves)
LibTelPtCnvrt()	Converts the FLASH data file pointer. (For Contacts.)
#LibFileWriteCheckInit()	Initializes LibFileWriteCheck().
#LibFileWriteCheck()	Checks whether data can be written to the FLASH memory.
LibFileReadEx()	Reads data form the FLASH memory.

■ Alarm Functions

Function name	Function
@ LibAlarm()	Calls an alarm process.
@ LibNextAlmSet()	Sets the Next Alarm.
LibChkSysAlarm()	Corrects the system alarm data.
*@ LibInitAlarmFlg()	Clears an alarm match flag.
*@ LibInitAlarmFlgCheck()	Checks whether the alarm time matched.
*@ LibNextAlarmSet()	Sets an alarm.
*LibSetDailyAlarm()	Sets a daily alarm time.
*@ LibInitAlarm()	Clears the alarm settings.
*@ LibGetAlarmInfo()	Gets the alarm status.
*@ LibGetAlarmFlg()	Gets the alarm flag.
*LibGetDailyAlarm()	Gets a daily alarm time.
*@ LibGetNextAlm()	Gets a next alarm pointer.
*LibAlarmBuzzSet()	Performs a buzzer setting during an alarm matches.
*@ LibGetAlarmObj()	Gets a touch table information as soon as the alarm time matched.

■ Date/Time Functions

Function name	Function
LibGetDateTimeM()	Gets the current date/time. Summer time correction is provided. (BCD 1 buffer)
LibGetDateTime()	Gets the current date/time. Summer time correction is provided. (BCD)
LibGetDateTime2()	Gets the current date/time. Summer time correction is provided. (Numeric number)
*LibGetDate()	Gets the system date. No summer time correction. (BCD)
*LibGetTime()	Gets the system time. No summer time correction. (BCD)
*LibGetDate2()	Gets the system date. No summer time correction. (Numeric number)
*LibGetTime2()	Gets the system time. No summer time correction. (Numeric number)
*LibAdjustTimeDeff2()	Corrects a date and time with the specified time lag.
LibChangeTotalDay()	Converts the specified number of days into year-month-day (in numeric format).
LibGetTotalDay2()	Gets the total number of days from the specified year-month-day (in numeric format).
LibSetDateTime()	Updates a date/time.
*LibSetDateTime2()	Updates a date/time.
*LibSetDate2()	Updates a date.
*LibSetTime2()	Updates a time.
LibGetDow()	Gets the day of the week from date.
LibGetDays()	Gets the number of days of the month from the year and month.
LibChkFuture()	Compares the size of the date/time data.(Old and new comparison.)
New!LibSummerTimeSet()	Set summer time
LibDateDisp()	Displays date string using the format.
LibWait()	Waits for a specified period of time.
LibCheckDate()	Tests validity of the date string..
LibChkTimeBuf()	Tests validity of the time string
LibClkDispLine()	Displays a time data.
LibClkDispCursor()	Displays a cursor for inputting a time.
New!LibConvRaw2Lib2()	Convert data in HHMM format into data for time input
LibConvRaw2Lib()	Converts the HHMM format data into data for inputting a time.
LibConvLib2Raw()	Converts a data for inputting a time into HHMM format data.
LibGetCursorPos()	Gets a cursor position on the time board from a touch location.
LibJumpDate()	Displays the date jump screen.

■ Character Input/Drag Event Functions

Function name	Function
LibTxtInit()	Initializes the general-purpose text input process.
LibTxtTchSet()	Registers a touch area for the general-purpose text input process.
LibTxtInp()	Performs the main processing of the general-purpose text input process.
LibTxtDsp()	Updates the display contents of the general-purpose text input process.
LibTxtKeyWordSet()	Performs the keyword registration of the general-purpose text input process.
LibTxtDspC()	Updates the display contents of the general-purpose data display process
LibTxtDspInit()	Initializes the general-purpose data display process.
LibTxtDspS()	Controls displays for the general-purpose data display process.
LibGetCursor()	Gets the cursor status.
LibCurBlnkOn()	Blinks a cursor.
LibCurBlnkOn2()	Blinks a cursor. (Blank type)
LibCurBlnkOff()	Turns OFF a cursor.
*LibCurErase()	Clears a cursor. (Compulsorily put a cursor in the off state.)
New! LibTxtWrapSw()	Specify indication using word wrap

■ Message Functions

Function name	Function
LibPutMessage()	Displays a built-in 5-language message.
LibPutMessageCenter()	Displays a built-in 5-language message. (With center justified.)
LibPutMessageCenter2()	Displays a built-in 5-language message. (With center position specified.)
LibPutMessageRight()	Displays a built-in 5-language message. (With right justified.)
LibReadMessage()	Reads a built-in 5-language message string.
*LibGetMessCnt()	Gets the number of lines for a built-in 5-language message.
*LibDspWinMessage()	Displays for a dialog message.
*LibGetWinMessSize()	Gets the position and size of the window for a dialog message.
LibErrorDisp()	Displays a message that corresponds to the error code related to the FLASH memory.

■ Character String Functions

Function name	Function
LibBCD2Ascii()	Converts BCD code to ASCII, and outputs it to the 2-byte buffer.
LibAscii2BCD()	Converts a 2-byte ASCII code to 1-byte BCD code.
LibNumoStr()	Converts a numeric number to character string.
LibStoNum()	Converts a character string to numeric number.
LibCuttextRtn()	Deletes the CR code attached to the end of a text or an item.
LibKeyWordInit()	Initializes the keyword registration area.
LibKeyWordSet()	Registers a character string to the keyword area.
LibKeyWordFSrch()	Performs the first keyword search..
LibKeyWordNSrch()	Performs the NEXT keyword search.
*LibKeyWordSrchSub()	Performs a search by the character string, and gets an appropriate keyword.
LibChangeBcdVal()	Converts a BCD code to numeric number.
LibChangeValBcd()	Converts a numeric number to BCD code.

■ Handwriting (INK) Functions

Function name	Function
LibDrawInit()	Initializes the drawing BIOS.
LibDrawSetPtn()	Specifies a pen contrast.
LibDrawSetClipArea()	Specifies a drawing area.
LibDrawSetPoint()	Draws a dot.
LibDrawLine()	Draws a line.
LibDrawBox()	Draws a box.
LibDrawCircle()	Draws a circle.
LibDrawFillArea()	Fills a rectangular area.
LibDrawTransDD()	Transfers the specified VRAM area to other VRAM area defined individually.
LibDrawTransAll()	Performs the entire screen data transfer between the specified virtual VRAMs, and between the specified system VRAMs.
LibDrawPutImage()	Writes an image to VRAM.
LibDrawGetImage()	Gets an image from VRAM.
LibDrawReductImage()	Reduces an image.
*LibDrawPrmCall()	Calls a drawing BIOS using the assigned function number.
LibScrShot()	Executes the screen-shot process.

■ Mode Functions

Function name	Function
LibJumpMenu()	Calls the MENU mode.
LibGetMode()	Gets various mode information.
LibDualWin()	Starts up the dual-window, and gets the data pointer for the dual side.
LibDualWinExit()	Quits the dual-window processing, and returns to the mode where the process was started up.
LibModeJump()	Jumps to the specified mode.
LibScrtJmp()	Jumps to the intermediate state for transiting to the secret mode.
LibSecretCall()	Calls the secret mode by the function specification.
LibScrtModeJmp()	Jumps from the intermediate state of the Secret mode transition to other mode.
LibCrdlOpnJmp()	Changes the mode to Open mode, and performs a forcible mode jump to PC link process.
LibMenuJump()	Jumps from MENU to other mode.
LibGetLastMode()	Gets the previous mode information.
LibDataCom()	Transits to the Data Communication process.
*LibCallListMenu()	Calls the list type menu.
LibPassWordCheck()	Checks the system password.
LibPassWordEdit()	Corrects the system password.
LibMoveArea()	Moves between Open area and Secret area.
LibModeRestart()	Restarts the current mode.
New! LibFileCom()	Run file transfer mode

■ Menu Functions

Function name	Function
LibSelWindow()	Displays the selection menu list.
LibSelWindow2()	Displays the selection menu list. (For Delete menu)
LibSelWindowExt()	Displays the selection menu list. Extended version.
LibSelWinExt2A()	Displays the selection menu list. (Only display.)
LibSelWinExt2B()	Displays the selection menu list. (Only wait for touching.)
LibWinIcnMsg()	Displays a general-purpose message dialog box.
LibPullDown()	Displays a general-purpose pull-down menu.
LibPullDownInit()	Makes the initial setting of the general-purpose pull-down menu.
LibPullDownAtrSet()	Makes the attribute setting of the general-purpose pull-down menu.
LibEditPullDown()	Performs the pull-down menu process during inputting data.
LibSelWinLckA()	Performs a window display process for the fixed message.
LibSelWinLckB()	Performs a list selection process for the fixed message.
LibSelectFont()	Performs a font selection window display process.
New! LibSelWinOpen2()	Open selection window
New! LibSelectWin()	Process item selection in window
New! LibSelWinTchSet()	Define touch area in selection window

■ System Functions

Function name	Function
LibSaveSysRam()	Saves all system area data for application to the FLASH memory.
LibSaveSysRamB()	Saves all system area data for BIOS to the FLASH memory.
LibGetBLD()	Checks the battery status.
LibGetVersion()	Gets the ROM creation version.
*LibELHandle()	Performs various EL-panel operations.
*LibGetEL()	Gets the EL-panel status.
*LibGetLang()	Gets the current language information.
*LibSetLang()	Sets/changes the system language information.
*LibSoundGet()	Gets the sound information.
*LibSoundSet()	Sets/changes the system sound information.
*LibContrastInit()	Initializes the contrast setting.
*LibContrastUp()	Adjusts the contrast setting one level darker.
*LibContrastDown()	Adjusts the contrast setting one level lighter.
*LibDigitizer()	Adjusts the touch-panel.
*LibPassClr()	Clears the system password.
*LibPassSet()	Sets/changes the system password.
*LibPassGet()	Gets the system password.
*LibPassChk()	Checks the system password.
*LibGetAPOTime()	Gets the APO time.
*LibSetAPOTime()	Sets the APO time.
*LibSetKeyKind()	Sets the keyboard layout type.
*LibGetKeyKind()	Gets the keyboard layout type.
*LibBuzzerOff()	Turn off a buzzer.
*LibBuzzerOn()	Turn on a buzzer.
*LibGetLangInf()	Gets the information whether the current ROM model is the single language version or the 5-language version.
New! LibGetCommDevice()	Get selected communication device
New! SysGetPONstat()	Get Touch PowerON setting information
New! SysSetPONstat()	Set Touch PowerON on/off
New! SysGetSUPstat()	Get opening specification screen display setting information
New! SysSetSUPstat()	Set opening specification screen display on/off
New! SysGetELTime()	Get EL backlight setting information
New! SysSetELTime()	Set length of EL backlight

■ Function Functions

Function name	Function
LibFuncDateTime()	Sets the date/time.
LibFuncSound()	Sets the sound information.
LibFuncFormat()	Sets various system formats.
LibFuncLang()	Sets the language.
LibFuncCapa()	Performs the check display of the FLASH capacity.
LibFuncContrast()	Sets the contrast.
LibFuncDigitizer()	Adjusts the touch-panel.
LibFuncMemoryManagement()	Calls the memory remaking process.
LibFuncPtool()	Calls the POP-UP-TOOL.
New! LibFuncUSB()	Select communication interface
LibCalWin()	Calls the calculator mode.

■ Calculator Functions

Function name	Function
LibCalBase()	Processes the four basic arithmetical calculation.
New! LibCalBase2()	Perform calculator operation (for original four-function and root operations)
LibCalRoot()	Processes the root calculation.
LibCalKeyInit()	Initializes the calculator keyboard.
LibCalKeyDsp()	Displays the calculator keyboard.
LibCalKeyTchWait()	Waits for touching of the calculator keyboard.

■ Debug Functions

Function name	Function
LibPutMsgDlg()	Displays a string formatted in conformity with printf(). (Wait for touching.)
LibPutMsgDlg2()	Displays a string formatted in conformity with printf(). (Wait for 0.5 seconds.)
LibPutMsgDlg3()	Displays a string formatted in conformity with printf(). (Wait for 0.125 seconds.)
LibPutMsgDlg4()	Displays a string formatted in conformity with printf(). (No wait for touching.)

■ ADD IN Functions

Function name	Function
LibFileFindNext()	Searches the FLASH memory for next data.
*@ LibExeAddin()	Executes the addin synchronization.
*LibGetDLAllNum()	Gets the total number of the Downloaded program or data.
*LibGetUserMode()	Searches and Gets the ModeCode and Status.
*LibGetProgramName()	Gets the Program name indicated the mode.
*LibGetModeVer()	Gets the Program version of the mode.
*LibGetLibVer()	Gets the Library version of the mode.
*LibGetMenuIcon()	Gets the Icon graphic for IconMenu.
*LibGetListIcon()	Gets the Icon graphic for ListMenu.
*LibCheckPMode()	Checks the existence of program.
LibSubEntrySave()	Saves the program file name(sub entry).
LibSubEntryDel()	Deletes the program file name(sub entry).
LibSubEntryRename()	Renames the program file name.
LibSubEntrySearch()	Searches the program file name.
LibGetSubEntrySt()	Gets the SubEntry status.
LibGetSubEntNum()	Gets the total number of SubEntry.
LibGetAllEntry()	Gets the MainEntry and SubEntry.

■ FLASH Functions (Call far pointer of the file buffer)

Function name	Function
@ LibLFileFindNext()........................	Searches the FLASH memory for next data.
@ LibLFileFindPrev()........................	Searches the FLASH memory for previous data.
@ LibLFileFindNextExt().................	Searches the FLASH memory for next data.(Extended version.)
@ LibLNextSearchCld().....................	Searches the FLASH memory for next data.(Only for Calendar)
@ LibLFileRead()	Reads data from the FLASH memory
@ LibLFileWrite()..............................	Writes data to the FLASH memory.
@ LibLFileCorect()............................	Corrects the FLASH data without changing the FLASH pointer.
@ LibLFileRemove()	Deletes data from the FLASH memory.(1 record)
@ LibLFileRemoveAll()	Deletes data from the FLASH memory.(All records in specified modes)
@ LibLGetFileInfo()	Gets the FLASH memory data information.
@ LibLGetFileCnt()............................	Gets the number of records in the specified mode registered in the FLASH memory.
@ LibLTodoFileRemove()..................	Deletes the TODO data.
@ LibLFileExch()	Sorts FLASH data. (Moves)
#@ LibLFileWriteCheck()	Checks whether data can be written to the FLASH memory.
@ LibLFileReadEx()..........................	Reads data form the FLASH memory.

■ Binary File Access Functions

Function name	Function
New!LibUbfFindFirst()	Retrieve binary file (first time only)
New!LibUbfFindNext()	Retrieve binary file (from second time onward)
New!LibUbfFindClose()	End binary file retrieval
New!LibUbfOpen()	Open binary file
New!LibUbfWrite()	Write data onto binary file
New!LibUbfRead()	Read data from binary file
New!LibUbfSeek()	Move file pointer to specified position
New!LibUbfClose()	Close binary file
New!LibUbfRemove()	Remove binary file
New!LibUbfRename()	Rename binary file
New!LibUbfLength()	Get size of currently opened binary file
New!LibUbfFlush()	Flush write buffer
New!LibUbfGetFree()	Get current free space in flash memory

■ Serial/USB Communication Functions

Function name	Function
LibSrlPortOpen()	Opens the serial/USB communications port.
LibSrlPortClose()	Closes the serial/USB communications port.
LibSrlPortFClose()	Compulsion closing of the serial communications port.
LibSrlRxBufClr()	Clears the receiving buffer.
LibSrlTxBufClr()	Clears the sending buffer.
LibSrlGetDteStat()	Gets DTE status.
LibSrl232CStat()	Gets status of RS232C signal line.
LibSrlRateSet()	Changes the DTE communication speed.
LibSrlGetRBufChar()	Gets number of data in the receiving buffer.
LibSrlGetTBufSpace()	Gets number of empty data in the sending buffer.
LibSrlSendByte()	Sends one byte.
LibSrlRecvByte()	Receives one byte.
LibSrlPreRead()	Previously reads the receiving buffer.
LibSrlSendBreak()	Sends a break signal.
LibSrlSendBlock()	Sends a block data.
LibSrlRecvBlock()	Receives a block data.
LibSrlGetOpenStat()	Gets the open status.