User's Manual

NEC

SM78K Series Ver. 2.30 or Later

System Simulator

External Part User Open Interface Specifications

Target Devices
78K/0 Series
78K/0S Series
78K/IV Series

Document No. U15802EJ1V0UM00 (1st edition)
Date Published February 2002 N CP(K)

[MEMO]

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Europe) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 01
Fax: 0211-65 03 327

- Branch The Netherlands
Eindhoven, The Netherlands
Tel: 040-244 58 45
Fax: 040-244 45 80

- Branch Sweden
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics (France) S.A.
Vélizy-Villacoublay, France
Tel: 01-3067-58-00
Fax: 01-3067-58-99

NEC Electronics (France) S.A.
Representación en España
Madrid, Spain
Tel: 091-504-27-87
Fax: 091-504-28-60

NEC Electronics Italiana S.R.L.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Guarulhos-SP, Brasil
Tel: 11-6462-6810
Fax: 11-6462-6829

J01.12

INTRODUCTION

Target Readers

The contents described in this manual use the Windows™ 95/Windows 98/Windows 2000/Windows NT™ 32-bit application program format and this manual is therefore intended for users who have experience creating Windows 95/Windows 98/Windows 2000/Windows NT 32-bit application programs.

Purpose

The purpose of this manual is to describe the interface specifications to enable users to create custom settings for standard external parts that cannot otherwise be used for the SM78K System Simulator. The functions, programming rules, and programming steps that users need to create programs for customized parts are described in this manual.

Organization

This manual is broadly divided into the following sections.

• General
• Download
• Programming
• Function reference
• Operations during CPU reset
• Programming examples
• Error messages

How to Use This Manual

It is assumed that readers of this manual have general knowledge of microcomputers and the C programming language. Readers will need to have a basic knowledge of how to create Windows 95/Windows 98/Windows 2000/Windows NT 32-bit application programs.

To find details of functions that can be used to create programs for customized parts:
→ See **CHAPTER 4 FUNCTION REFERENCE**.

To understand the meanings and causes of messages:
→ See **APPENDIX A ERROR MESSAGES**.

Target Products

The "SM78K" described in this manual represents the following products.

Product Name	Supporting Series
SM78K0	8-bit single-ship microcontroller 78K/0 Series (except for small-scale general-purpose products)
SM78K0S	8-bit single-chip microcontroller 78K/0S Series (small-scale general-purpose products)
SM78K4	16-bit single-chip microcontroller 78K/IV Series

Also, the description "78KX" in this manual should be replaced as follows according to the system simulator used.

System Simulator Used	Description in This Manual	Actual Name
SM78K0	78KX	78K0
SM78K0S	78KX	78K0S
SM78K4	78KX	78K4

Example Replace “SU78KX.DLL” as follows.
For 78K0: SU78K0.DLL
For 78K0S: SU78K0S.DLL
For 78K4: SU78K4.DLL

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Note:	Footnote for item marked with **Note** in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numerical representation:	Binary … XXXX or XXXXB
	Decimal ... XXXX
	Hexadecimal … 0xXXXX

Prefix indicating the power of 2 (address space, memory capacity):

K (Kilo): $2^{10} = 1024$
M (Mega): $2^{20} = 1024^2$

Related Documents Refer to the documents listed below when using this manual.
The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents related to 78K Series development tools (user's manuals)

Document Name		Document No.
CC78K0S C Compiler Ver. 2.30 or Later	Operation	U14871E
	Language	U14872E
CC78K0 C Compiler Ver. 3.30 or Later	Operation	U14297E
	Language	U14298E
CC78K4 C Compiler Ver. 2.20 or Later	Operation	U15557E
	Language	U15556E
RA78K0S Assembler Package	Operation	U14876E
	Language	U14877E
	Structured Assembly Language	U11623E
RA78K0 Assembler Package	Operation	U14445E
	Language	U14446E
	Structured Assembly Language	U11789E
RA78K4 Assembler Package	Operation	U15254E
	Language	U15255E
	Structured Assembler Preprocessor	U11743E
SM78K Series System Simulator Ver. 2.30 or Later	Operation (Windows Based)	U15373E
SM78K Series System Simulator Ver. 2.30 or Later	External Part User Open Interface Specifications	This manual
ID78K Series Integrated Debugger Ver. 2.30 or Later	Operation (Windows Based)	U15185E
RX78K0 Real-Time OS	Fundamental	U11537E
	Installation	U11536E
RX78K4 Real-Time OS	Fundamental	U10603E
	Installation	U10604E
Project Manager Ver. 3.12 or Later (Windows Based)		U14610E

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 GENERAL

1.1 General Description of External Part User Open Interface Specifications

In addition to simulating the operations of the actual target system, the SM78KX can simulate the operations of a dummy target system.

Standard external parts are provided with the SM78KX for building a dummy target system. Setup dialog boxes are also provided for each external part to enable easier implementation of standard external parts.

In addition, parts that cannot be set up using a setup dialog box for standard external parts still can be implemented via user programming as user-specified external parts.

The external part user open interface specifications include the function specifications for the SM78KX's interface, which the user needs to create programs for customized parts.

1.2 General Description of User-Customized Parts

1.2.1 Types of customization

Parts can be customized by the user's programming in the following two ways.

(1) Customization via Parts window

Parts can be customized using the customization function that facilitates the creation of parts by simply giving the relevant pins and action information as parameters.

Based on information that is called within a user's function, the corresponding part is pasted into the Parts window and all of the related simulation processing is executed.

(2) Customization via user window

Users can customize parts with functions that can be used to create parts and windows.

The handle notification function for a user window can be used to enable processing of windows and input from user parts, and the simulation call function can be used to perform output display processing to user parts.

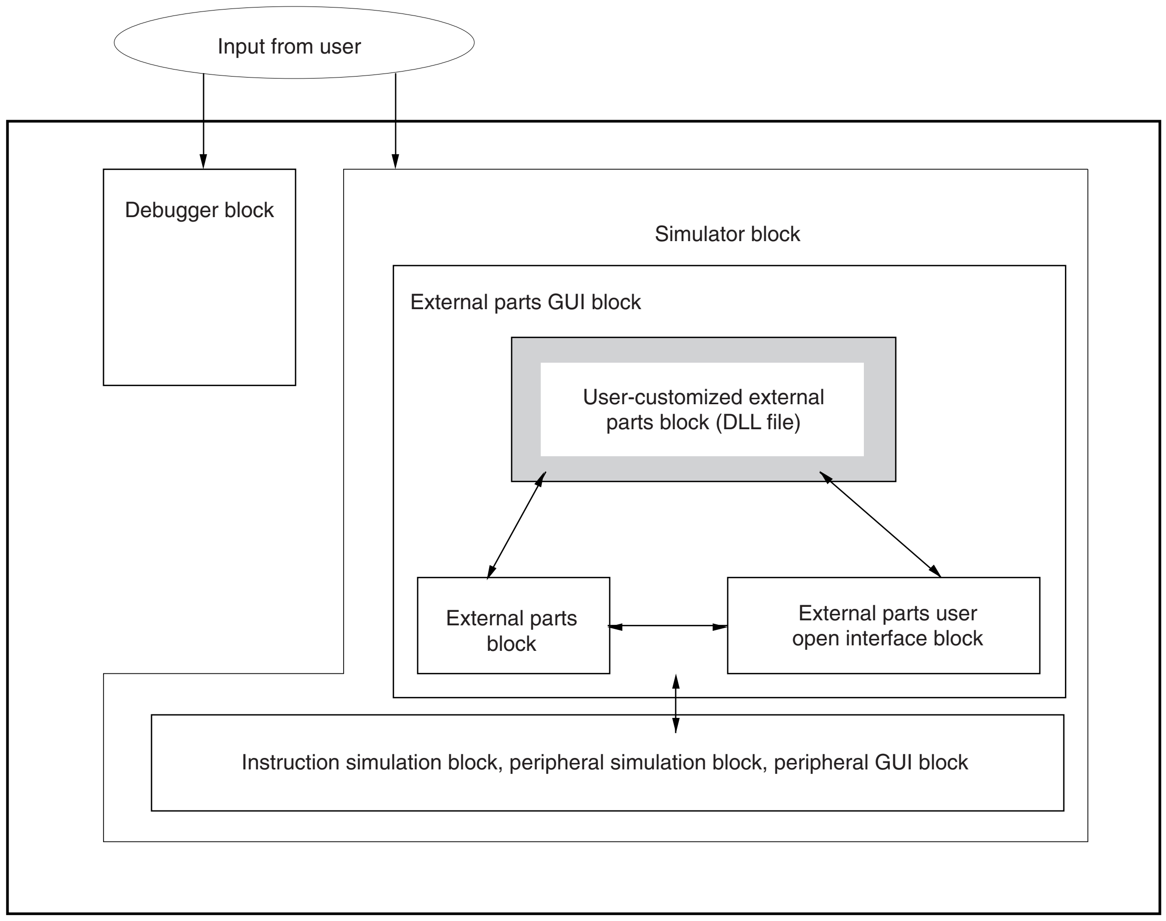
1.2.2 User-created files

User-customized parts are implemented by user-created programs based on the specifications described in this manual. These user-created programs end up as DLL files.

The DLL files for user-customized parts are loaded into the external parts GUI block before simulation processing is executed.

1.2.3 Positioning of user-customized parts

Figure 1-1. Configuration Diagram of 78KX Series Simulator


Input from user
Debugger block
Simulator block
External parts GUI block
User-customized external parts block (DLL file)
External parts block
External parts user open interface block
Instruction simulation block, peripheral simulation block, peripheral GUI block

Debugger block	Any directive from the user that causes any function to be executed by the simulator is called a command. The debugger block provides an environment in which the user can enter such commands via the keyboard or the mouse.
Peripheral GUI block	This block provides a setup environment that enables the user to easily set the desired input information to a port via a window.
DLL	DLL stands for "Dynamic Link Library." DLLs are Windows modules that contain executable code and data that can be accessed by functions within Windows applications or other DLLs.
External parts GUI block	This block enables external part operations to be performed via a window.
External parts block	This is part of the external parts GUI block, and is used to control standard external parts.
User-customized external parts block	This is part of the external parts GUI block, and is used for user-created external parts.
External parts user open interface block	This is part of the external parts GUI block, and is used as an interface between the external parts block and the user-customized external parts block.

1.3 Environment

1.3.1 Development environment

The following describes the development environment under which users write programs according to this manual's specifications in order to create DLL files.

Hardware environment:	NEC PC-9821/PC98-NX Series, IBM PC/AT™ compatible
	(CPU: Pentium™ 120 MHz or above is recommended)
Software environment:	Windows 95/Windows 98/Windows 2000/Windows NT 4.0
	Microsoft Visual C++ V5.00 or later

1.3.2 Operating environment

The operating environment of the simulator that loads and operates user-created files is described below.

Hardware environment:	NEC PC-9821/PC98-NX Series, IBM PC/AT compatible
	(CPU: Pentium 120 MHz or above is recommended)
Software environment:	Windows 95/Windows 98/Windows 2000/Windows NT 4.0

1.4 Cautions When Transferring External Parts Created for SM78K Series Ver. 1.42 or Earlier to Ver. 2.30 or Later

To use the external parts created for the Ver. 1.42 or earlier versions of the SM78K Series in the Ver. 2.30 or later, part of the external part source needs to be modified and the external part recreated.

Here, the modified parts are described.

1.4.1 Change of source

C source is modified as follows.

- Change the file to be included from uparts.h to uparts32.h.
  uparts32.h is in .\smp78kx\sm78kx under the SM78K installation directory (e.g. c:\nectools32).
- Use DllMain() instead of the LibMain() or WEP() functions.

1.4.2 Change of make environment

A new Win32 make environment needs to be created. When creating a make environment in VC++, note the following two points.

(1) Add the uparts32.cpp file to the project.
uparts32.cpp is in .\smp78kx\sm78kx under the SM78K installation directory (e.g. c:\nectools32).

(2) Set the single-byte alignment of structure members.

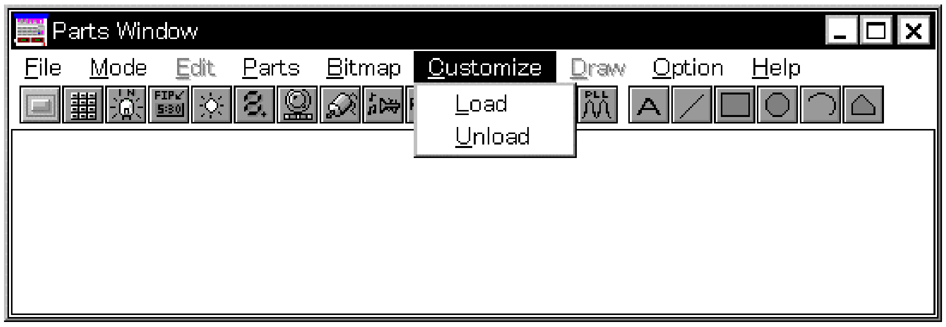
CHAPTER 2 DOWNLOAD

This chapter describes the steps for downloading to the simulator user-customized parts that are created as described in Chapters 3 and 4.

Before user-customized external parts (DLL files) can be actually used, they must be loaded into the simulator. To remove loaded user-customized external parts (DLL files), unload them from the simulator.

Use the Parts window to load and unload user-customized external parts (DLL files).

Figure 2-1. SM78KX Simulator Parts Window

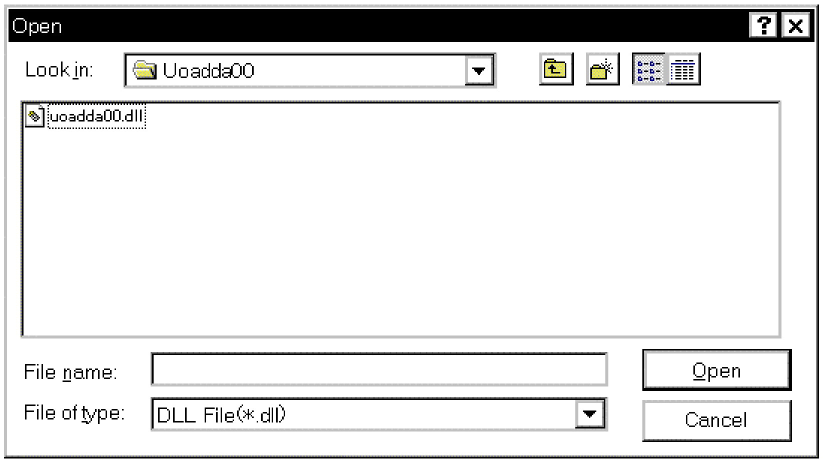


2.1 Download

Operation steps

(1) In the Parts window, select [Customize] → [Load] from the menu bar to open the Open dialog box.

Figure 2-2. Open Dialog Box



(2) In the Open dialog box, select a customized external part DLL file, then click the <Open> button. The specified DLL file is then loaded into the simulator. Once this has been done, the part created by the customization function in the Parts window is pasted in the Parts window. If the part was customized via a user window, it is displayed in a user window.

(a) Up to six user-customized external part DLLs can be loaded into the simulator.
(b) A user-customized external part DLL file that is downloaded to the simulator remains valid even after the Parts window is closed. The next time the Parts window is opened, the same DLL file will be automatically downloaded.
(c) The name of the loaded user-customized external part DLL file is added to the pull-down menu under the [Customize] menu of the Parts window.
(d) The user-customized external part that is displayed in the Parts window can be relocated. However, the information about the relocation cannot be saved. After relocation, if either of the following sets of operations have been performed, the location of each part is neither saved nor completed. Therefore, be sure to locate each part again.
- If the status is saved to a project file (xxxx.prj) or to a file to which display information for the Parts window is to be saved (xxxx.pnl), and then these files are read
- If the Parts window is closed while customized external part DLL information remains loaded, and then the Parts window is opened again

2.2 Unload

Operation steps

(1) Select [Customize] → [Unload] from the menu bar in the Parts window.
(2) This unloads (removes) all of the customized external part DLLs that are currently loaded in the simulator. Parts that have been created by the Parts window's customization function are deleted from the Parts window. Also, if there are any programs that have been customized via a user window, the user window is closed.

CHAPTER 3 PROGRAMMING

3.1 Programming Configuration and Processing Flow

This chapter describes the basic programming used for customization via the Parts window and customization via a user window.

3.1.1 Customization via Parts window

Configuration

The configuration includes user functions that are called only once after the DllMain function (required to create DLL files) and the DLL files have been loaded.

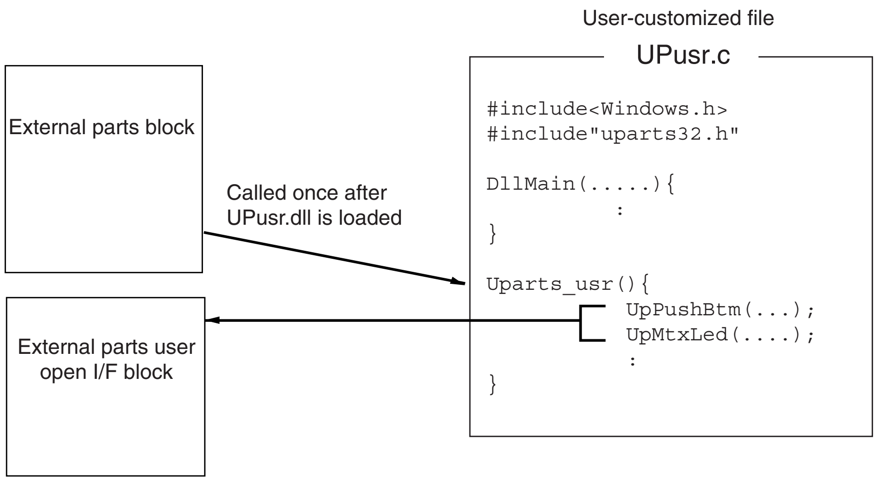
Function references described in Chapter 4 must be included either in user functions or in functions subordinate to user functions.

Processing flow

The simulator's external parts block is used to create parts based on the specified function's part information and performs all simulation related to parts associated with the simulator's external parts block.

Figure 3-1 shows the relationship between user-created DLL files and external parts in the simulator, as well as the configuration of functions.

Figure 3-1. Programming Configuration and Processing Flow for Customization via Parts Window


User-customized file
UPusr.c
External parts block
Called once after UPusr.dll is loaded
External parts user open I/F block

```
#include<Windows.h>
#include"uparts32.h"

DllMain(.....){
          :
}

Uparts_usr(){
          UpPushBtm(...);
          UpMtxLed(....);
          :
}
```

3.1.2 Customization via user window

Configuration

The configuration includes the DllMain function (required to create DLL files), the created window's callback functions, user functions, and simulation call functions that are called at a set interval during simulations.

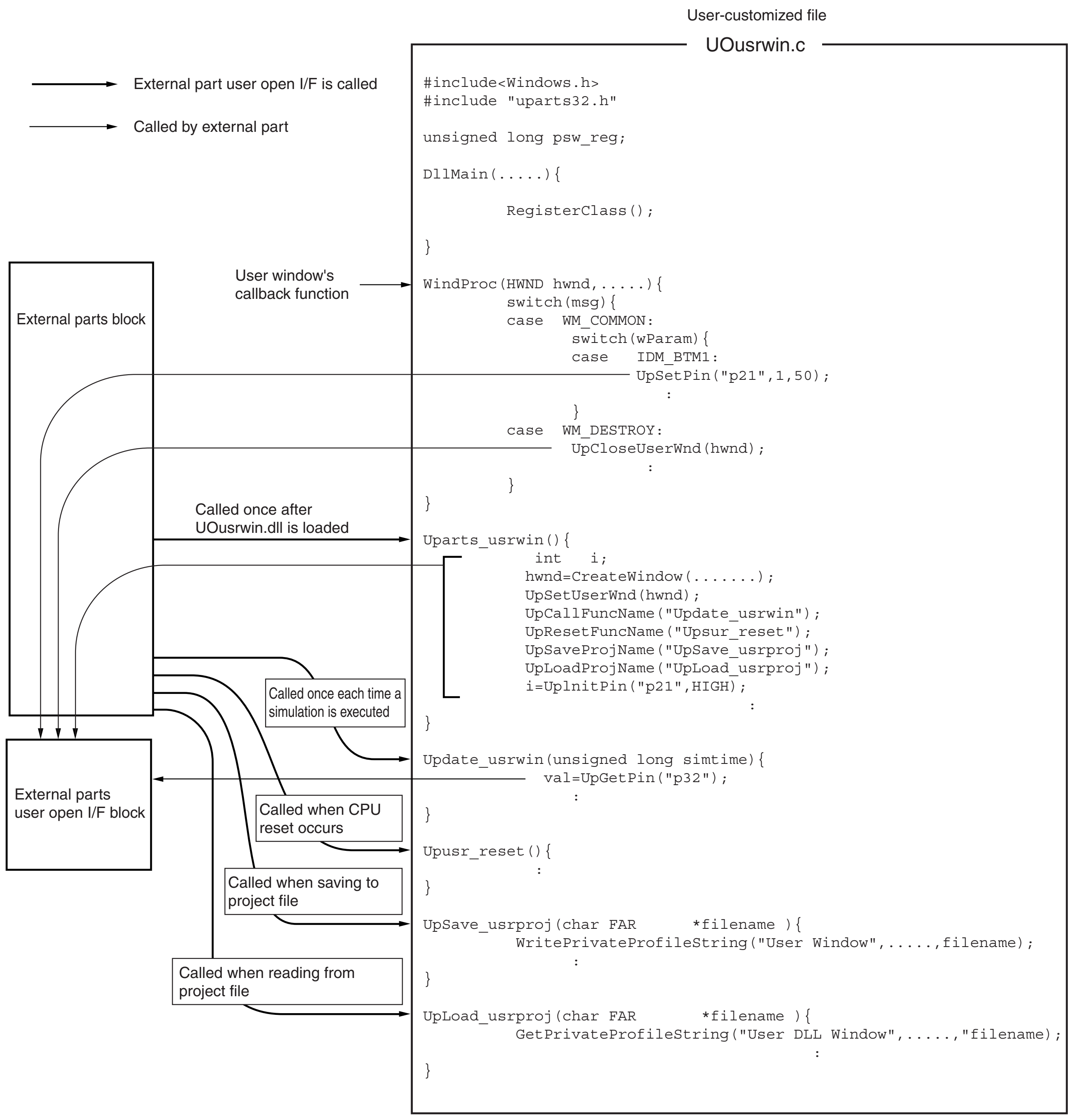
User functions and their subordinate functions are used to report simulation call functions and the motor pin names. The creation of parts and programming of I/O actions are done using the user-created window's callback functions and simulation call functions.

Processing flow

Simulation of customized parts is performed as the simulator works with the external parts block using functions that capture and set I/O information on pins and ports. The pin output information also can be redrawn (or otherwise processed) by calling simulation call functions from the external parts block.

Figure 3-2 shows the relationship between DLL files customized via a user-created window and external parts in the simulator, as well as the configuration of functions.

Figure 3-2. Programming Configuration and Processing Flow for Customization via User Window


User-customized file
UOusrwin.c
External part user open I/F is called
Called by external part
External parts block
User window's callback function
Called once after UOusrwin.dll is loaded
Called once each time a simulation is executed
External parts user open I/F block
Called when CPU reset occurs
Called when saving to project file
Called when reading from project file

```
#include<Windows.h>
#include "uparts32.h"

unsigned long psw_reg;

DllMain(.....){

         RegisterClass();

}

WindProc(HWND hwnd,.....){
         switch(msg){
         case  WM_COMMON:
                switch(wParam){
                case   IDM_BTM1:
                       UpSetPin("p21",1,50);
                          :
                }
         case  WM_DESTROY:
                UpCloseUserWnd(hwnd);
                         :
         }
}

Uparts_usrwin(){
            int   i;
           hwnd=CreateWindow(.......);
           UpSetUserWnd(hwnd);
           UpCallFuncName("Update_usrwin");
           UpResetFuncName("Upsur_reset");
           UpSaveProjName("UpSave_usrproj");
           UpLoadProjName("UpLoad_usrproj");
           i=UplnitPin("p21",HIGH);
                                     :
}

Update_usrwin(unsigned long simtime){
             val=UpGetPin("p32");
                 :
}

Upusr_reset(){
            :
}

UpSave_usrproj(char FAR      *filename ){
          WritePrivateProfileString("User Window",.....,filename);
                 :
}

UpLoad_usrproj(char FAR       *filename ){
          GetPrivateProfileString("User DLL Window",.....,"filename);
                                                  :
}
```

3.2 Steps in Creation of Customized Parts

3.2.1 Customization via Parts window

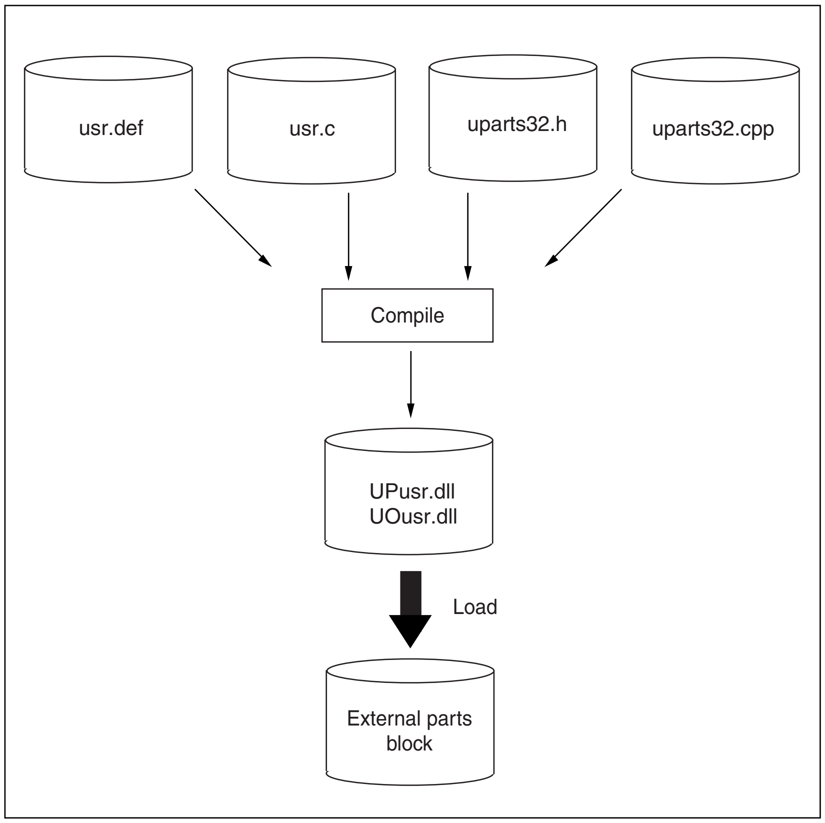
1. Program the external parts to be customized when creating a DLL file using Windows programming methods. Be sure to include the file "uparts32.h" in this programming and add “uparts32.cpp” to the project.
2. Use Windows programming methods to create a module definition (DEF) file$^{\text{Note}}$, a make file, and, if necessary, a resource file, then compile to create a user-created DLL file.
   - When compiling, specify the option (/Zp1) for single-byte alignment of structure members.
   - Specify "UP" as the first two characters in the name of the created DLL file.
   - To operate the DLL file in an environment in which Microsoft Visual C++ is not installed, create the DLL file using the released version.
3. Enter the user-created DLL file name in the place for specifying the simulator's external parts customization files (See **2.1 Download**).
4. In addition to the standard parts that are already displayed in the Parts window, the user-created customized parts are displayed.
5. Set the Parts window to location mode and locate the parts.
6. Select [Save As...] from the [File] menu of the Parts window and save the current status so that there will not be any need to load the user-created DLL files when performing the next simulation.

3.2.2 Customization via user window

1. Program the external parts to be customized when creating a DLL file using Windows programming methods. Be sure to include the file "uparts32.h" in this programming and add “uparts32.cpp” to the project.
2. Use Windows programming methods to create a module definition (DEF) file$^{\text{Note}}$, a make file, and, if necessary, a resource file, then compile to create a user-created DLL file.
   - When compiling, specify the option (/Zp1) for single-byte alignment of structure members.
   - Specify "UO" as the first two characters in the name of the created DLL file.
   - To operate the DLL file in an environment in which Microsoft Visual C++ is not installed, create the DLL file using the released version.
3. Enter the user-created DLL file name in the place for specifying the simulator's external parts customization files (See **2.1 Download**).
4. The window created by the user and the corresponding customized parts are displayed.

Note See **3.4 Module Definition (DEF) File**.

Figure 3-3. Creation Flow



3.3 Basic Rules

The basic rules for user programming of customized parts are described below.

3.3.1 User functions

User functions are main functions that are described by users.

(1) When a user-created DLL file is loaded to the simulator, it becomes a function that is called by the simulator.
(2) The function references described in Chapter 4 must be contained in user functions or functions that are subordinate to user functions.
(3) User function names are function names in which the name of the user-created DLL file minus the first two characters is added to "UParts_".
(4) The first two characters of the user-created DLL file name are fixed.
    (a) Customization via Parts window
        Always use "UP" as the first two characters of the user-created DLL file name.
        Example: UPusr.dll → UParts_usr()
    (b) Customization via user window
        Always use "UO" as the first two characters of the user-created DLL file name.
        Example: UOusr.dll → UParts_usr()
(5) Use void type with no parameters for user functions.
(6) Enter an EXPORTS declaration[Note] in the module definition file for user functions.

Note See **3.4.1 EXPORTS declaration**.

3.3.2 External variables

When using external variables, always add "UP" to the start.

Example: int UPglobal

3.3.3 Function names

Function names are the names that are given to user-created external parts.

If a function name need not be specified as the part name, enter a NULL string as the parameter of the function used to create the part.

3.3.4 Active high/low

The "active high/low" designation specifies the relation between a pin's value and its active state (when a part connected to a pin is operating). If the function used to create a part includes a parameter for specifying "active high/low," specify one of the following macros (the macros "HIGH" and "LOW" are defined in uparts32.h).

Operation using 1 (high):	HIGH
Operation using 0 (low):	LOW

3.3.5 Pin names

Some of the parameters in functions used to create parts are for specifying pin names or port names. In such cases, each pin name or port name is specified as a character string, and its name should be as described in the target device's user's manual. Specifications are not case sensitive.

3.3.6 Include file, Source file

The include file “uparts32.h” and source file “uparts32.cpp” that are used for user customization are bundled in the SM78K product package. Include uparts32.h and link uparts32.cpp.

uparts32.h and uparts32.cpp are in . \smp78kx\sm78kx under the SM78K installation directory (e.g. c:\nectools32).

- uparts32.h contains descriptions of macro definitions for active high/low status, and IMPORTS declarations for functions described in **CHAPTER 4 FUNCTION REFERENCE**.
- When compiling, be sure to set the include path in the directory where the file "uparts32.h" is located.

3.4 Module Definition (DEF) File

A module definition (DEF) file must be created to include the EXPORTS declaration, as described in the module definition file for Windows programming.

The IMPORTS declaration is already included in the file "Uparts32.h" and therefore does not need to be considered.

3.4.1 EXPORTS declaration

Be sure to enter an EXPORTS declaration for user functions and simulation call functions.

EXPORTS declarations must be entered for functions used to read or save project files, for reset functions, and some other functions.

Example:
```
EXPORTS    UParts_usrwin
           UPdata_usrwin
```

CHAPTER 4 FUNCTION REFERENCE

4.1 Customization via Parts window

The functions that can be pasted into the Parts window to perform all simulation processing by simply calling within the user functions are listed below.

These functions can be used to easily create parts simply by specifying pins and action information as parameters.

Note, however, that even if the user has created a window, all parts that are created by this function are still pasted in the Parts window.

Table 4-1. Customization Functions Used in Parts Window

Function Name	Prototype	Page
Push-button function	UpPushBtm(*pname, actype, btmname*)	26
Toggle button function	UpTglBtm(*pname, actype, btmname*)	27
Group select button (exclusive push-button) function	UpSelectBtm(*gname, pnames, pnum, actype, btmnames*)	28
Hold time setup function	UpSetPBtmtime(*time*)	30
LED function	UpLed(*pname, actype, ledname, pictype*)	31
LED function set per port	UpPortLed(*portname, actype, ledname, pictype*)	33
Matrix LED function	UpMtxLed(*pnames1, pnames2, pnum1, pnum2, actype1, actype2*)	35
DC motor function	UpDcMtr(*pname, actype, mtrname*)	37
Stepper motor function	UpStpingMtr(*pnames, num, actype, reiji, step*)	38
Vertical scroll bar analog input function	UpScaleInterAD(*pname, adname*)	40
Reference voltage value setup function	UpSetAVref(*avref*)	42
Bitmap setup function for button	UpSetBtmBmp(*actbmp, nactbmp*)	43
Bitmap setup function for LED	UpSetLedBmp(*actbmp, nactbmp*)	44
Bitmap setup function for DC motor	UpSetMtrBmp(*actbmp, nactbmp*)	45
LED picture setup function	UpSetLedPic(*type, color*)	46
Serial pin data input function	UpSerial_data(*serpname, data, count, first*, *bitnum*)	47
Window title function	UpPanelTitleName(*title*)	48
Bitmap display function	UpSetUsrBmp(*bmpname*)	49
Character string display function	UpWriteString(*string*)	50

Push button function

```
void    UpPushBtm(pname, actype, btmname)
char    *pname;          /* Pin name */
int     actype;          /* Active high/low */
char    *btmname;        /* Function name */
```

[Function]

This function creates one push button. A push button is a button icon that sets and holds the input status for a specified hold time only after the button has been clicked. The hold time is set using the hold time setup function UpSetPBtmtime().

The time set in UpSetPBtmtime described before this function is assumed as the hold time. If a hold time is not set, the default value of 0.5 ms is used.

[Parameters]

pname	Specifies the pin name as a character string.
actype	Specifies a value to be input using a push button. Specify HIGH to enter a "1" (high value) or LOW to enter a "0" (low value).
btmname	Specifies the name of the push button function. Since this function name is displayed on the button, the character string is limited to 16 single-byte characters.

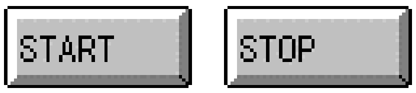
[Return value]

None

[Example]

```
UpSetPBtmtime(50);
UpPushBtm("p20",HIGH,"START");
UpPushBtm("p20",LOW,"STOP");
```

Figure 4-1. Push Buttons



Toggle button function

```
void    UpTglBtm(pname, actype, btmname)
char    *pname;         /* Pin name */
int     actype;         /* Active high/low */
char    *btmname;       /* Function name */
```

[Function]

This function creates one toggle button. When clicked, a toggle button sets and holds the input status until the same button is clicked again.

This button's initial mode is inactive. The first time this button is clicked, the value specified by the parameter *actype* is input.

[Parameters]

pname	Specifies the pin name as a character string.
actype	Specifies a value to be input using the toggle button. Specify HIGH to enter a "1" (high value) or LOW to enter a "0" (low value).
btmname	Specifies the name of the toggle button function. Since this function name is displayed on the button, the character string is limited to 16 single-byte characters.

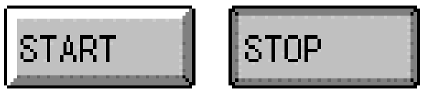
[Return value]

None

[Example]

```
UpTglBtm("p22",HIGH,"START");
UpTglBtm("p23",LOW,"STOP");
```

Figure 4-2. Toggle Buttons



Group select button (exclusive push button) function

```
void    UpSelectBtm(gname, pnames, pnum, actype, btmnames)
char    *gname;                 /* Group name */
char    **pnames;               /* Pin name */
int     pnum;                   /* Number of buttons */
int     actype;                 /* Active high/low */
char    **btmnames;             /* Function name */
```

[Function]

Several buttons can be grouped together as exclusive buttons. Clicking one of the group of buttons enclosed in a frame enters an active value for the clicked button only.

The entered value remains in effect until another button is clicked. In other words, there can be only one active button at a time within the button group.

[Parameters]

gname	Specifies the name assigned to the group. This group name is shown at the top of the group select buttons.
pnames	Specifies pin names (character strings) for each button.
pnum	Specifies the number of buttons.
actype	Specifies the value entered by clicking a group select button. Specify HIGH to enter a "1" (high value) or LOW to enter a "0" (low value). The active status for all group buttons is the same.
btmnames	Specifies the names assigned to individual buttons. Since this function name is displayed on the button, the character string is limited to 10 single-byte characters.

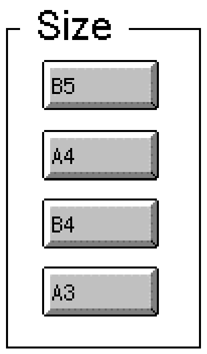
[Return value]

None

[Example]

```
static  char  *sizePin[4] =   {"p30","p31","p32","p33"};
static  char  *sizeName[4] =  {"B5","A4","B4","A3"};
UpSelectBtm("Size",  sizePin,  4,  HIGH,  sizeName);
```

Figure 4-3. Group Select Buttons



Hold time setup function

```
void    UpSetPBtmtime(time)
char    *time;              /* Hold time */
```

[Function]

This function specifies the hold time for a push button.

[Parameter]

time	Sets a hold time character string. The unit for this setting is ms (milliseconds). The range of settings is 0.001 to 999 ms.

[Return value]

None

[Example]

```
UpSetPBtmtime("0.2");
```

LED function

void	UpLed(*pname, actype, ledname, pictype*)	
char	**pname*;	/* Pin name */
int	*actype*;	/* Active high/low */
char	**ledname*;	/* Function name */
char	*pictype*;	/* Picture type */

[Function]

This function creates one LED.

When the specified pin's status is active, an active bitmap (or color picture) is displayed. When the pin's status is inactive, an inactive bitmap (or colorless picture) is displayed.

[Parameters]

pname	Specifies the pin name as a character string.
actype	Specifies the value to be displayed on the LED. Specify HIGH for active high or LOW for active low.
ledname	Specifies the LED's function name. This function name is shown on the LED. There is no limit on the number of characters.
pictype	Specifies the type of picture (or bitmap image) used in the LED display. If 1: The default bitmap type is a light bulb-type bitmap image. However, any bitmap specified by the UpSetLedBmp() function is displayed instead of the default bitmap. If 0: The default picture type is a rectangular picture. Any picture specified by the UpSetLedPic() function is displayed instead of the default picture.

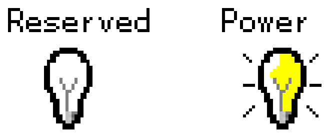
[Return value]

None

[Example]

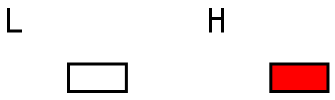
```
UpLed("p40",LOW,"Reserved",1);
UpLed("p21",HIGH,"Power",1);
```

Figure 4-4. Bitmap Images for Inactive LED (Left) and Active LED (Right)



```
UpLed("p41",LOW,"L",0);
UpLed("p22",HIGH,"H",0);
```

Figure 4-5. Pictures for Inactive LED (Left) and Active LED (Right)



LED function set per port

void	UpPortLed(*portname, actype, ledname, pictype*)	
char	**portname*;	/* Port name */
unsigned	char *actype*;	/* Active high/low */
char	**ledname*;	/* Function name */
char	*pictype*;	/* Picture type */

[Function]

This function creates a set of LEDs corresponding to pins assigned to a particular port (eight LEDs make one set). An active bitmap (or color picture) is displayed for each pin that is active and an inactive bitmap (or colorless picture) is displayed for each pin that is inactive.

[Parameters]

portname	Specifies the port name as a character string.
actype	Specifies the value for displaying an active bitmap. Specify "1" if a value of "1" (high) is active or specify "0" if a value of "0" (low) is active. The 8-bit data that sets the status of eight LEDs is specified bitwise. Values are specified bitwise for 8 bits, starting from the port's lowest pin as the LSB. **Example** When p30 and p31 are active low for port 3's LED and all other pins are active high set the lower 2 bits of actype to 0: UpPortLed("p3", 0xfc, "Number", 1);
ledname	Specifies a name to be assigned to an LED. This function name is shown below the bitmap. There is no limit on the number of characters.
pictype	Specifies the picture type used in the LED display. "1" specifies bitmap and "0" specifies a rectangular picture.

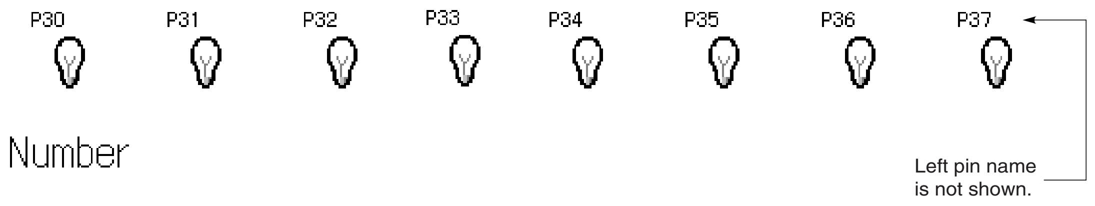
[Return value]

None

[Example]

```
UpPortLed("p3",0xfc,"Number",1);
```

Figure 4-6. LED Function Set Per Port



Matrix LED function

```
void    UpMtxLed(pnames1, pnames2, pnum1, pnum2, actype1, actype2)
char    **pnames1;        /* Output 1 pin names */
char    **pnames2;        /* Output 2 pin names */
int     pnum1;            /* Output 1 pin number */
int     pnum2;            /* Output 2 pin number */
int     actype1;          /* Active high/low for output 1*/
int     actype2;          /* Active high/low for output 2*/
```

[Function]

This function creates an LED on a matrix. When any intersection is active on the matrix of the output 1 and output 2 pins, a matrix LED showing the active bitmap is created (the active bitmap is fixed and cannot be specified).

[Parameters]

pnames1	Specifies the output 1 pin names (character strings) for all output 1 pins only.
pnames2	Specifies the output 2 pin names (character strings) for all output 2 pins only.
pnum1	Specifies the number of output 1 pins.
pnum2	Specifies the number of output 2 pins.
actype1	Specifies the value for displaying output 1. Specify HIGH for active high status or LOW for active low status. The active status for output 1 is the same for all output 1 pins.
actype2	Specifies the value for displaying output 2. Specify "HIGH" for active high status or "LOW" for active low status. The active status for output 2 is the same for all output 2 pins.

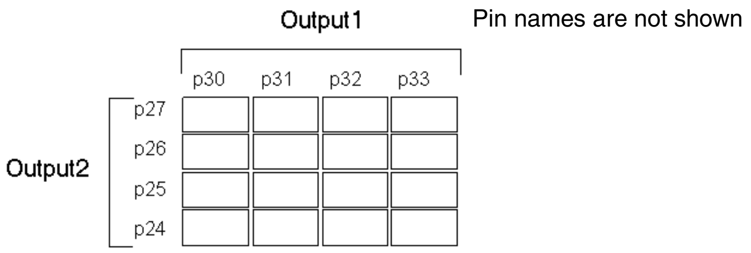
[Return value]

None

[Example]

```
static  char  *out1[4] = {"p30","p31","p32","p33"};
static  char  *out2[4] = {"p24","p25","p26","p27"};
UpMtxLed(out1, out2, 4, 4, HIGH, HIGH);
```

Figure 4-7. Matrix LED Function



DC motor function

```
void    UpDcMtr(pname, actype, mtrname)
char    *pname;          /* Pin name */
int     actype;          /* Active high/low */
char    *mtrname;        /* Function name */
```

[Function]

This function creates a DC motor icon. An active bitmap is displayed when the specified pin becomes active, and an inactive bitmap is displayed when the specified pin is inactive.
This function also displays the total active time that has elapsed since the start of a simulation. The displayed time is based on the main system clock. When a reset occurs or when the elapsed time value exceeds a 10-digit decimal value, the displayed time is cleared to zero.

[Parameters]

pname	Specifies the pin name as a character string.
actype	Specifies the status when the motor is displayed as active. Specify HIGH for active high status or LOW for active low status.
mtrname	Specifies the DC motor function's name. This function name is shown under the motor icon. There is no limit on the number of characters.

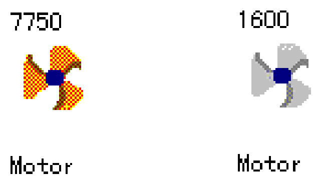
[Return value]

None

[Example]

UpDcMtr("p41",HIGH,"Motor");

Figure 4-8. Active LED (Left) and Inactive LED (Right)



Stepper motor function

```
void   UpStpingMtr(pnames,num,actype,,reiji,step)
char   **pnames;          /* Pin names */
int    num;               /* Number of pins per channel */
int    actype;            /* Active high/low */
char   reiji;             /* Excitation method */
short  step;              /* Minimum step angle */
```

[Function]

This function creates a stepper motor that is operated via several pins.
The motor is displayed according to its direction of rotation, with the rotation speed and step angles.

[Parameters]

pname	Specifies pin names (character strings) for all pins.
num	Specifies the number of pins per channel (4 or 8).
actype	Specifies the status when the motor is displayed as active. Specify HIGH for active high status or LOW for active low status. The active status is the same for all pins.
reiji	Specifies the excitation method. Set "0" for single phase or "1" for single/dual phase.
step	Specifies an integer fraction of 360 as the minimum step angle.

[Return value]

None

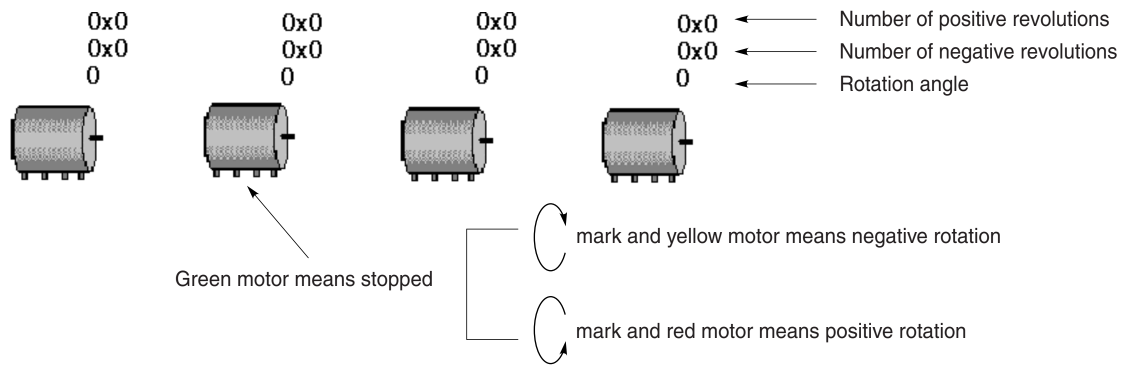
[Remarks]

Once operation of this function is started, the first value other than zero that is output to a connected pin is taken as the initial value. At that point, the stepper motor is shown as stopped (not rotating).

[Example]

```
char  *mtrpin[4] = {"p00","p01","p02","p03"};
UpStpingMtr(mtrpin, 4,HIGH,1,10);
```

Figure 4-9. Stepper Motor


0x0
0x0
0
Number of positive revolutions
Number of negative revolutions
Rotation angle
Green motor means stopped
mark and yellow motor means negative rotation
mark and red motor means positive rotation

Vertical scroll bar analog input function

```
void    UpScaleInterAD(pname, adname)
char    *pname;            /* Pin name */
char    *adname;           /* Function name */
```

[Function]

This function creates an analog input part for a vertical scroll bar.

Move the scroll box and right-click the mouse over the scroll bar to enable input of analog data. Input values are used to create a part that is shown in red.

[Parameters]

pname	Specifies the name of an analog input pin as a character string.
adname	Specifies the function name of the scroll-bar-type input part. This function name is displayed above the scroll-bar-type input part and there is no limit on the number of characters.

[Return value]

None

[Remarks]

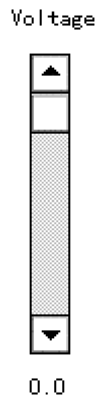
The scroll bar's operating range is determined either by settings made via the reference voltage value setup function UpSetAVref() or by the reference voltage value settings made via the Standard Level Gauge Pin Setting dialog box[Note]. If neither of these settings have been made, the default value of 5.0 V is used.

Note See **CHAPTER 6 WINDOW REFERENCE** in the **SM78K Series System Simulator Ver.2.30 or later Operation (Windows Based) (U15373E)**.

[Example]

```
UpScaleInterAD("ani1","Voltage");
```

Figure 4-10. Vertical Scroll Bar Analog Input



Reference voltage value setup function

```
void    UpSetAVref(avref)
char    *avref;              /* Reference voltage value */
```

[Function]

This function sets the reference voltage value for the A/D converter.
This reference voltage value is used to determine the operating range for an analog input part.
Any setting that is within the range for the operating power supply voltage (see the user's manual of each device) can be set.
Values can be set to the first decimal place, with subsequent decimal places rounded off.

[Parameter]

avref	Specifies the reference voltage value as a character string.

[Return value]

None

[Remarks]

If this function or the standard setting is not set, the analog input part will operate using the default voltage value of 5.0 V.

[Example]

```
UpSetAVref("3.5");
```

Bitmap setup function for button

```
void    UpSetBtmBmp(actbmp,nactbmp)
char    *actbmp;        /* Active bitmap name character string */
char    *nactbmp;       /* Inactive bitmap name character string */
```

[Function]

This function sets the bitmap for a button. The button display can be changed by entering this function immediately before the target button's function. The same bitmap will be displayed until it is set again by this function. The bitmap file should be stored in the same directory as the simulator or its name should be specified with the full path.

[Parameters]

actbmp	Specifies a character string for the bitmap file name displayed when active.
nactbmp	Specifies a character string for the bitmap file name displayed when inactive.

[Return value]

None

[Remarks]

If a button function is described without describing this function first, the standard button's bitmap is displayed (see the image shown in **Figure 4-1**). The button name is not shown when setting this function.

[Example]

```
UpSetBtmBmp("on.bmp","off.bmp");
UpPushBtm("p21",LOW,"START");
```

Bitmap setup function for LED

```
void    UpSetLedBmp(actbmp,nactbmp)
char    *actbmp;         /* Active bitmap name character string */
char    *nactbmp;        /* Inactive bitmap name character string */
```

[Function]

This function sets the bitmap for an LED. The LED display can be changed by entering this function immediately before the target LED's function. This function is valid only if the bitmap has been specified by an LED function. The same bitmap will be displayed until it is set again by this function. The bitmap file should be stored in the same directory as the simulator or its name should be specified with the full path.

[Parameters]

actbmp	Specifies a character string for the bitmap file name displayed when active.
nactbmp	Specifies a character string for the bitmap file name displayed when inactive.

[Return value]

None

[Remarks]

If an LED function is described without describing this function first, the standard LED's bitmap is displayed (see the image shown in **Figure 4-4**).

[Example]

```
UpSetLedBmp("lighton.bmp","lightoff.bmp");
UpLed("p31",HIGH,"Power",1);
```

Bitmap setup function for DC motor

```
void    UpSetMtrBmp(actbmp,nactbmp)
char    *actbmp;         /* Active bitmap name character string */
char    *nactbmp;        /* Inactive bitmap name character string */
```

[Function]

This function sets the bitmap for a DC motor. The DC motor display can be changed by entering this function immediately before the target DC motor's function. The same bitmap will be displayed until it is set again by this function. The bitmap file should be stored in the same directory as the simulator or its name should be specified with the full path.

[Parameters]

actbmp	Specifies a character string for the bitmap file name displayed when active.
nactbmp	Specifies a character string for the bitmap file name displayed when inactive.

[Return value]

None

[Remarks]

If a DC motor function is described without describing this function first, the standard DC motor's bitmap is displayed (see the image shown in **Figure 4-8**).

[Example]

```
UpSetMtrBmp("trun.bmp","stop.bmp");
UpDcMtr("p32",HIGH,"Motor");
```

LED picture setup function

```
void   UpSetLedPic(type,color)
char   type;                      /* Picture type */
char   color;                     /* Picture fill color when active */
```

[Function]

This function sets the type of picture and fill color (when active) to be used in an LED display. The LED display can be changed by entering this function immediately before the target LED's function. This function is valid only if a picture has been specified by an LED function. The same picture will be displayed until it is set again by this function.

[Parameters]

type	Specifies the type of picture (macro is defined in uparts32.h). Macro PIC_RECT: Rectangle Macro PIC_ELL: Ellipse
color	Specifies fill color when active (macro is defined in uparts32.h). Macro PIC_RED: Red Macro PIC_YELLOW: Yellow Macro PIC_GREEN: Green

[Return value]

None

[Example]

```
UpSetLedPic(PIC_RECT,PIC_GREEN);
UpLed("p32",HIGH,"Test",0);
```

Serial pin data input function

```
void      UpSerial_data(serpname,data,count,first,bitnum)
char      *serpname;             /* Serial pin name character string */
unsigned  short   *data;         /* Pointer to data array */
unsigned  short   count;         /* Number of data arrays */
char      first;                 /* First bit (MSB or LSB)*/
char      bitnum;                /* Number of bits in transfer data */
```

[Function]

This function sets values in order starting from the specified first data bit, using the number of bits in the data transferred to the serial pin as one unit.

[Parameters]

serpname	Specifies the character string for the name of the serial data input pin.
data	Specifies a pointer to an array in which the value set to the serial data input pin has been stored in units consisting of the number of transfer data bits.
count	Specifies the number of arrays in which values set to the serial data input pin have been stored in units consisting of the number of transfer data bits.
first	Specifies whether data equivalent to the number of bits in the transfer data will be set sequentially with the MSB first or the LSB first. Specify "1" to set sequentially with the MSB first and specify "0" to set sequentially with the LSB first.
bitnum	Specifies the number of bits in the transfer data. When using UART (Universal Asynchronous Receiver/Transmitter), the start bit, parity bit, and stop bit are included in the data and data bit count.

[Return value]

None

[Example]

To set 8-bit data sequentially from LSB first to serial pin SER1:

```
unsigned short data[3] = {0xff, 0xa0, 0x3b};
UpSerial_data("SER1",data,3,0,8);
```

The data is input to SER1 as shown below.

```
← 111111110000010111011100
```

Window title function

```
void    UpPanelTitleName(title)
char    *title;             /* Title name */
```

[Function]

This function displays a name in the title bar of the Parts window.

[Parameter]

title	Specifies the character string for the name to be displayed in the title bar of the Parts window.

[Return value]

None

[Example]

```
UpPanelTitleName("System for printer");
```

Bitmap display function

```
void    UpSetUsrBmp(bmpname)
char    *bmpname;        /* Bitmap file name */
```

[Function]

This function displays a bitmap that is always displayed, unrelated to simulations.

The bitmap is displayed to the right of the part that is at the bottom right in the set of currently displayed parts. If there is not enough room in the window to display the bitmap to the right of the bottom right part, it is displayed below the bottom right part.

[Parameter]

bmpname	Specifies a character string as the bitmap file name. Bitmap file names should be specified using the file names in the same directory as the simulator or with the full path.

[Return value]

None

[**Example**]

```
UpSetUsrBmp("printer.bmp");
```

Character string display function

```
void    UpWriteString(string)
char    *string;              /* Character string to be displayed */
```

[Function]

This function displays a character string.

The character string is displayed to the right of the part that is at the bottom right in the set of currently displayed parts. If there is not enough room in the window to display the character string to the right of the bottom right part, it is displayed below the bottom right part.

[Parameter]

string	Specifies the character string to be displayed. There is no limit on the number of characters.

[Return value]

None

[Example]

```
UpWriteString("Power");
```

4.2 Customization via User Window

The following functions are provided to enable the user to freely customize user-created windows and parts. The handle notification function for user windows can be used to enable processing of windows and input from user parts, and the simulation call function can be used to perform output display processing to user parts.

Table 4-2. Customization Functions Used in User Window

Function Name	Prototype	Page
Window handle notification function	UpSetUserWnd(*hUwnd*)	52
Window close function	UpCloseUserWnd(*hwnd*)	53
Simulation call function	UpCallFuncName(*fname*)	54
Motor pin notification function	UpInitMtrPin(*pname,actype*)	55
Stepper motor notification function	UpInitStpingMtr(*pname, num, actype, reiji, step*)	56
Pin active value notification function	UpInitPin(*pname, actype*)	57
Port active value notification function	UpInitPort(*portname, actype*)	58
AD input pin notification function	UpInitAD(*pname*)	59
Project file read function name notification function	UpLoadProjName(*funcname*)	60
Project file save function name notification function	UpSaveProjName(*funcname*)	61
Reset function name notification function	UpResetFuncName(*funcname*)	62
Pin value capture function	UpGetPin(*pname, val*)	63
Port data capture function	UpGetPort(*portname, data*)	64
DA output pin value capture function	UpGetDA(*pname, val*)	65
Memory area data capture function	UpGetMem(*addr, data*)	66
DC motor active time clear function	UpClrMtrAcClk(*pname*)	67
Stepper motor information capture function	UpGetStpingMtr(*pnames, num, posrev, negrev, angle*)	68
Control register data capture function	UpGetReg (*type*)	69
Value setting function for pins	UpSetPin(*pname, val, time*)	70
Data setting function for ports	UpSetPort(*portname, data, time*)	71
Value setting function for AD input pin	UpSetAD(*pname, val*)	72
Data setting function for memory area	UpSetMem(*addr, data*)	73
Active time notification function for motor	UpGetMtrAcClk(*pname, val, actime*)	74
Data setting function for control register	UpSetReg (*type, data*)	75
Time conversion notification for one main system clock pulse	UpSimtimeSec(*void*)	76
Function for transmitting packets from HOST using USB function	UpSetUSBPack(*total, total_bit, data*)	77
Function for receiving packets from Function using USB function	UpGetUSBPack(*total, data*)	78
Function for transmitting signals from HOST using USB function	UpSetUSBSig(*sig*)	79
Function for receiving signals from Function using USB function	UpGetUSBSig(*sig*)	80

Window handle notification function

```
void      UpSetUserWnd(hUwnd)
HANDLE    hUwnd;          /* Handle of user window */
```

[Function]

This function notifies the simulator of a user-created window handle.
The user should describe this function immediately after creating a window.

[Parameter]

hUwnd	Handle of a user-created window

[Return value]

None

[Example]

```
HWND    hwnd;
hwnd = CreateWindow(........);
UpSetUserWnd(hwnd);
```

Window close function

```
void    UpCloseUserWnd(hwnd)
HWND    hwnd;              /* Handle of window to be closed */
```

[Function]

This function notifies the simulator that a user-created window is being closed.
This function is described with the user-created window callback function's message WM_DESTROY.

[Parameter]

hwnd	Handle of user-created window to be closed

[Return value]

None

[Example]

```
WM_DESTROY:
    :
    :
UpCloseUserWnd(hwnd);
```

Simulation call function

```
void    UpCallFuncName(fname)
char    *fname;             /* Simulation call function name */
```

[Function]

This function reports the name of the function that is called from the simulator at a specified interval[Note] during simulation.
This function must be described within the user function UParts_xxx().

Note This function is called once per command execution.

[Parameter]

fname	Specifies the name of the function called from the simulator.

[Return value]

None

[Remarks]

The simulation call function should be specified as follows in the function specifications.
The simulation's execution time is received via an unsigned long type parameter.
The simulation's execution time is time that has elapsed since the previous function call, and its measurement unit is the main system clock.
Be sure to enter an EXPORTS declaration (see **3.4.1 EXPORTS declaration**) in a module definition file for the simulation call function.

```
void  FAR  PASCAL  Update_usrwin(unsigned long simtime)
```

[Example]

```
UpCallFuncName("Update_usrwin");
```

Motor pin notification function

```
void    UpInitMtrPin(pname,actype)
char    *pname;          /* Pin name */
int     actype;          /* Active high/low */
```

[Function]

This function reports the pin name specified for the motor to capture the motor value and active time via the active time notification function for motor.
When using the motor pin, this function must be described within the user function UParts_xxx().
When not using the motor pin, there is no need to describe this function.

[Parameters]

pname	Specifies a character string as the pin name connected to the motor.
actype	Specifies that the motor is in active mode. Specify HIGH for active high status or LOW for active low status.

[Return value]

None

[Remarks]

Unless notification is already included in the user function, even if the information is captured by the active time notification function for motor UpGetMtrAcClk() during a simulation, the captured value is not guaranteed.

[Example]

```
UpInitMtrPin("p41",HIGH)
```

Stepper motor notification function

```
int     UpInitStpingMtr(pnames,num,actype,reiji,step)
char    **pnames;       /* Pin name */
int     num;            /* Number of pins per channel */
int     actype;         /* Active high/low */
char    reiji;          /* Excitation method */
short   step;           /* Minimum step angle */
```

[Function]

This function connects a stepper motor that is operated via several pins to the specified pin.
When using the stepper motor, this function must be described within the user function UParts_xxx(). When not using the stepper motor, there is no need to describe this function.

[Parameters]

pnames	Specifies pin names (character strings) for all pins.
num	Specifies the number of pins per channel (4 or 8).
actype	Specifies the status when the motor is displayed as active. Specify HIGH for active high status or LOW for active low status. The active status is the same for all pins.
reiji	Specifies the excitation method. Set "0" for single phase or "1" for single/dual phase.
step	Specifies an integer fraction of 360 as the minimum step angle.

[Return value]

If set correctly:	1
If not set correctly:	0

[Example]

```
char  *mtrpin[4] = {"p00","p01","p02","p03"};
UpInitStpingMtr(mtrpin, 4,HIGH,1,10);
```

Pin active value notification function

```
int     UpInitPin(pname,actype)
char    *pname;          /* Pin name */
int     actype;          /* Active value of pin */
```

[Function]

This function sets the active mode value for one pin.
When there is a value to be input for a pin, this function must be described within the user function UParts_xxx().
When there is no value to be input for a pin, there is no need to describe this function.

[Parameters]

pname	Specifies the pin name as a character string.
actype	Specifies the active value of a pin. Specify HIGH for active high status or LOW for active low status.

[Return value]

If pin's active value was set correctly:	1
If pin's active value was not set correctly:	0

[Example]

When set to operate when the pin P46 is active high (when the input value = 1):

```
int  ret;
ret=UpInitPin("P46",HIGH);
```

Port active value notification function

int	UpInitPort(*portname,actype*)		
char	**portname*;		/* Port name */
unsigned	char	*actype*;	/* Active value of port */

[Function]

This function sets the active mode value for one port.
When there is a value to be input for a port, this function must be described within the user function UParts_xxx().
When there is no value to be input for a port, there is no need to describe this function.

[Parameters]

portname	Specifies the port name as a character string.
actype	Specifies the active value for each pin of a port. Specify "1" for port pins that have active high status or "0" for port pins that have active low status. Values are specified bitwise for 8 bits, starting from the port's lowest pin as the LSB.

[Return value]

If port's active value was set correctly:	1
If port's active value was not set correctly:	0

[Example]

When port 4's pins P40 and P41 are set as active high and pins P42 to P47 are set as active low:

```
int  ret;
ret=UpInitPort(“P4”,0x03);
```

When port 2's pin P27 only is set as active high and pins P20 to P26 are set as active low:

```
int  ret;
ret=UpInitPort(“P2”,0x80);
```

AD input pin notification function

```
int     UpInitAD(pname)
char    *pname;          /* AD input pin name */
```

[Function]

This function notifies the simulator of the AD input pin used to input a value from the user open interface function. If UpSetAD() includes a value to be input to the AD input pin, this function must be described within the user function UParts_xxx(). When the user open interface function does not include a value to be input to the AD input pin, there is no need to describe this function.

[Parameter]

pname	Specifies the AD input pin name as a character string.

[Return value]

Normal end:	1
Abnormal end:	0 (if the AD input pin does not exist in a device used in the current simulation)

[Example]

```
UpInitAD(“ANI0");
```

Project file read function name notification function

```
void    UpLoadProjName(funcname)
char    *funcname;        /* Project file read function name */
```

[Function]

When the simulator's project file is being read, this function reports the name of the function that simultaneously reads the information in the user window from the project file.
This function must be described within the user function UParts_xxx().

[Parameter]

funcname	Specifies the name of the function that reads the project file that has been called from the simulator.

[Return value]

None

[Remarks]

The project file read function's specifications are as follows.

- The project file name character string is received via the char FAR* type parameter.
- User window information is also read from the file named by the project file name that was received via the parameter. At that time, select either the GetPrivateProfileString or GetPrivateProfileInt function for the library used in the read operation.
- The section name used by the user is "User DLL Window".
- An EXPORTS declaration is required in a module definition file for the project file read function.
  void FAR PASCAL UpLoad_usrproj(char FAR *filename)

[Example]

UpLoadProjName(“UpLoad_usrproj”);

Project file save function name notification function

```
void    UpSaveProjName(funcname)
char    *funcname;         /* Project file save function name */
```

[Function]

When the simulator's project file is being saved, this function reports the name of the function that simultaneously saves the information in the user window to the project file.
This function must be described within the user function UParts_xxx().

[Parameter]

funcname	Specifies the name of the function that saves the project file that has been called from the simulator.

[Return value]

None

[Remarks]

The project file save function's specifications are as follows.

- The project file name character string is received via the char FAR* type parameter.
- User window information is also written to the file named by the project file name that was received via the parameter. At that time, select the WritePrivateProfileString function for the library used in the write operation.
- The section name used by the user is "User DLL Window".
- An EXPORTS declaration is required in a module definition file for the project file save function.
  void FAR PASCAL UpSave_usrproj(char FAR *filename)

[Example]

```
UpSaveProjName("UpSave_usrproj");
```

Reset function name notification function

```
void    UpResetFuncName(funcname)
char    *funcname;         /* Reset function name */
```

[Function]

When a CPU reset is called by the simulator, this function reports the function name that is used for the user window's reset processing.
This function must be described within the user function UParts_xxx().

[Parameter]

funcname	Specifies the name of the reset function called by the simulator.

[Return value]

None

[Remarks]

The reset function's specifications are as follows.

- It is a VOID type function since it has no parameters.
- An EXPORTS declaration is required in a module definition file for the reset function.
  Void FAR PASCAL Upreset_usrwin(VOID)

[Example]

```
UpResetFunName("Upreset_usrwin");
```

Pin value capture function

```
int     UpGetPin(pname,val)
char    *pname;          /* Pin name */
char    *val;            /* Pointer to area where pin value is stored */
```

[Function]

This function captures the value for one pin.

[Parameters]

pname	Specifies the pin name as a character string.
val	Specifies a pointer to the area where the pin value is stored.

[Return value]

If pin value was successfully captured:	1
If pin value was not successfully captured[Note]:	0

Note "0" is also returned if the pin value is undefined.

[Example]

```
char   val;
int    ret;
ret  = UpGetPin(“p46",&val);
```

Port data capture function

```
int         UpGetPort(portname, data)
char        *portname;          /* Port name */
unsigned    char      *data;    /* Pointer to area where port data is stored */
```

[Function]

This function captures port data.

[Parameters]

portname	Specifies the port name as a character string.
data	Specifies a pointer to the area where the port data is stored.

[Return value]

If port data was successfully captured:	1
If port data was not successfully captured[Note]:	0

Note "0" is also returned if the port values include any undefined values.

[Example]

```
unsigned   char   data;
int   ret;
ret  = UpGetPort(“p4",&data);
```

DA output pin value capture function

int	UpGetDA(*pname,val*)		
char	**pname*;		/* DA output pin name */
unsigned	short	**val*;	/* DA output value */

[Function]

This function sets the value of the DA output pin.

[Parameters]

pname	Specifies the DA output pin name as a character string.
val	Specifies a pointer to the area where the value of the DA output pin is stored.

[Return value]

Normal end:	1
Abnormal end[Note]:	0

Note "0" is also returned if the value of the DA output pin is undefined.

[Example]

```
unsigned short daval;
UpGetDA(“ANO0",&daval);
```

Memory area data capture function

```
int        UpGetMem(addr,data)
unsigned   long     addr;          /* Address */
unsigned   char     *data;         /* Data storage area */
```

[Function]

This function captures the data in the memory area.

[Parameters]

addr	Specifies an address in the memory area to be captured.
data	Specifies the data storage area.

[Return value]

If data was successfully captured:	1
If data was not successfully captured:	0

[Example]

```
unsigned  char   data;
int   ret;
ret = UpGetMem(0xffe000,&data);
```

DC motor active time clear function

```
VOID   UpClrMtrAcClk(pname)
char   *pname;          /* Pin name */
```

[Function]

This function zero-clears the active time of the specified motor-connected pin.

[Parameter]

pname	Specifies the motor-connected pin name as a character string.

[Return value]

None

[Remarks]

When using this function, call the motor pin notification function UpInitMtrPin() from within a user function so that the pin name is reported in advance.

[Example]

```
UpClrMtrAcClk(“p41");
```

Stepper motor information capture function

```
int          UpGetStpingMtr(pnames,num,posrev,negrev,angle)
char         **pnames;              /* Pin names */
int          num;                   /* Number of pins per channel (specify 4 or 8)*/
unsigned     long    *posrev;       /* Area for storing the number of positive revolutions */
unsigned     long    *negrev;       /* Area for storing the number of negative revolutions */
unsigned     long    *angle;        /* Area for storing angle */
```

[Function]

This function captures the number of positive/negative revolutions and current angle of the stepper motor that is connected to the pin names previously reported by the stepper motor notification function UpInitStpingMtr.

[Parameters]

pnames	Specifies pin names (character strings) for all pins.
num	Specifies the number of pins per channel (4 or 8).
posrev	Specifies the area where the number of positive revolutions is stored.
negrev	Specifies the area where the number of negative revolutions is stored.
angle	Specifies the area where the angle is stored.

[Return value]

If successfully captured:	1
If not successfully captured:	0

[Example]

```
char  *mtrpin[4] = {"p00","p01","p02","p03"};
unsigned  long   posrev;
unsigned  long   negrev;
unsigned  long   angle;
UpInitStpingMtr(mtrpin, 4,HIGH,1,10);
        :
UpGetStpingMtr(mtrpin,4,&posrev,&negrev,&angle);
```

Control register data capture function

```
unsigned        long    UpGetReg (type)
unsigned        char    type;               /* Value to identify control register */
```

[Function]

This function captures data of the control register (PSW, PC, or SP).

[Parameter]

type	Specifies the identification value of the control register to which data need to be set. Specify using a macro. PSW → Macro REG_PSW PC → Macro REG_PC SP → Macro REG_SP

[Return value]

Values of the specified control register.
When the PC is specified, the return value is the address of the next instruction executed.
When the PSW or SP is specified, it is the result of the current instruction execution.

[Example]

```
unsigned    long    psw-val;
psw-val = UpGetReg (REG_PSW);
```

Value setting function for pins

```
void       UpSetPin(pname,val,time)
char       *pname;                  /* Pin name */
char       val;                     /* Active value */
unsigned   long   time;             /* Hold time */
```

[Function]

This function sets a pin value.

[Parameters]

pname	Specifies the pin name as a character string.
val	Sets value when pin is active.
time	Sets a time for holding data. The time measurement unit is the main system clock.

[Return value]

None

[Remarks]

When using this function, call the pin active value notification function UpInitPin() from within a user function so that the pin name is reported in advance. If the pin active value that is reported by UpInitPin was set as macro HIGH, setting "1" as the active value for this UpSetPin function sets the pin to active mode. Similarly, if the pin active value that is reported by UpInitPin was set as macro LOW, setting "0" as the active value for this UpSetPin function sets the pin to active mode. If "0" is set for the hold time, the active value is held.

[Example]

If UpInitPin("p31",HIGH) is described and the pin P31 is reported as active high, the description shown below sets the active high input to be held for 50 pulses of the main system clock.

```
char  val;        val = 1;
UpSetPin(“p31”,val,50L);
```

Data setting function for ports

```
void        UpSetPort(portname,data,time)
char        *portname;              /* Port name */
unsigned    char    data;           /* Data */
unsigned    long    time;           /* Hold time */
```

[Function]

This function sets the data in port units.

[Parameters]

portname	Specifies the port name as a character string.
data	Specifies values set to the port.
time	Sets a time for holding data. The time measurement unit is the main system clock.

[Return value]

None

[Remarks]

When using this function, call the port active value notification function UpInitPort() from within a user function so that the pin names are reported in advance. If the active value of the port's pins reported by UpInitPort was set as active high, "1" is set bitwise, and if it was set as active low, "0" is set bitwise. If pins belonging to this port are set to active mode by this UpSetPort function, the data's bit values for the corresponding pins should be the same as the bit values corresponding to the pins whose active values were set by UpSetPort.
If "0" is set for the hold time, the active value is held.

[Example]

If UpInitPort("p4",0x03) is described and port P4's pins P40 and P41 are reported as active high while pins P42 to P47 are reported as active low, the description shown below sets port P4's pins P40, P42, and P43 to active mode and holds the active mode for 50 pulses of the main system clock.

```
unsigned  char  data;
data = 0xf1;
UpSetPort(“p4",data,50L);
```

Value setting function for AD input pin

```
int         UpSetAD(pname,val)
char        *pname;          /* AD input pin name */
unsigned    short    val;    /* AD input value */
```

[Function]

This function sets the value of the AD input pin.

[Parameters]

pname	Specifies AD input pin name as a character string.
val	Sets value to be input to AD input pin.

[Return value]

Normal end: 1
Abnormal end[Note]: 0

Note "0" is returned if the AD input pin does not exist in a device used in the current simulation.

[Remarks]

When using this function, the AD input pin connection notification function UpInitAD() must be called from within a user function so that the AD input pin name is reported in advance.

[Example]

```
unsigned short adval;

adval = 10;
UpSetAD(“ANI0",adval);
```

Data setting function for memory area

```
int         UpSetMem(addr,data)
unsigned    long   addr;            /* Address */
unsigned    char   data;            /* Data */
```

[Function]

This function sets data in a memory area.

[Parameters]

addr	Specifies an address in the target memory area.
data	Specifies data.

[Return value]

If value is set correctly:	1
If value is not set correctly:	0

[Example]

```
int  ret;
ret = UpSetMem(0xffe300, 0x72);
```

Active time notification function for motor

```
int          UpGetMtrAcClk(pname, val, actime)
char         *pname;                     /* Pin name */
char         *val;                       /* Value */
unsigned     long       *actime;         /* Active time */
```

[Function]

This function captures the active time of the pin specified for a motor.

This function is valid only for pins connected to a motor part that has already been created using the motor pin notification function UpInitMtrPin().

The active time is the total time that has elapsed since the start of a simulation. When a reset occurs or when the elapsed time value exceeds a 10-digit decimal value, the active time is cleared to zero.

The active time is measured in pulses of the main system clock.

[Parameters]

pname	Specifies the motor-connected pin name as a character string.
val	Sets the value of the pin.
actime	Uses a two-dimensional array to represent the active time as the total time that has elapsed since the start of a simulation. actime[1]×0x100000000+actime[0] **Example**: actime[1] = 0x390; actime[0] = 0x10052688; Total time = 0x39010052688 main system clock

[Return value]

If set pin was a pin set by <u>DC motor function</u>:	0
If set pin was not a pin set by <u>DC motor function</u>:	−1

[Remarks]

When using this function, call the motor pin notification function UpInitMtrPin() from within a user function so that the pin name is reported in advance.

[Example]

```
char    val;
unsigned  long  actime[2];
UpGetMtrAcClk("p41",&val,actime);
wsprintf(timebuf,"Rotation time =   #%lx%08lx \n",actime[1],actime[0]);
TextOut(hdc,240,320,timebuf,sizeof(timebuf));
```

Data setting function for control register

```
void       UpGetReg (type, data)
unsigned   char    type;     /* Value to identify control register */
unsigned   long    data;     /* Data */
```

[Function]

This function sets data to the control register (PSW, PC, or SP).

[Parameters]

type	Specifies the identification value of the control register to which data needs to be set. Specify using a macro. PSW → Macro REG_PSW PC → Macro REG_PC SP → Macro REG_SP
data	Specifies the data to be set to the specified control register. Note that parts that exceed the register size are omitted.

[Return value]

None

[Remarks]

The control register data set by this function is set at the end of the first simulation. Therefore, if the control register value is rewritten during the simulation of the first instruction (when simulation is performed several times by the instruction) using this function, the control register value may change depending on the rest of the simulation results. To set data per instruction simulation, data of the control register needs to be rewritten at the timing when the PC value obtained by the user using the UpGetReg function is changed.

[Example]

```
UpSetReg(REG_PSW, 0x2);
```

Time conversion notification for one main system clock pulse

```
unsigned        long    UpSimtimeSec(void)
```

[Function]

This function converts one pulse of the main system clock to a nanosecond value.

[Parameter]

None

[Return value]

The nanosecond value converted from one pulse of the main system clock is returned.

[Example]

```
unsigned long simtime;
simtime = UpSimtimeSec( );
```

Function for transmitting packets from HOST using USB function

```
BOOL        UpSetUSBPack(total,total_bit,data)
unsigned    char    total;              /* Number of data arrays */
unsigned    char    total_bit;          /* Number of bits in transmit data */
unsigned    char    *data;              /* Pointer to packet data array */
```

[Function]

This function uses the USB function to set packet transmission information from the HOST.

[Parameters]

total	Specifies the number of packet data arrays.
total_bit	Specifies the total number of bits in the data to be transmitted.
data	Specifies a pointer to the packet data array to be transmitted.

[Return value]

Normal end:	1
Abnormal end:	0

[Remarks]

This function is supported only for devices that include a USB function.

Function for receiving packets from Function using USB function

```
void        UpGetUSBPack(total,data)
unsigned    char    total;              /* Number of data arrays */
unsigned    char    **data;             /* Pointer to packet data array */
```

[Function]

This function uses the USB function to receive packet data from Function.

[Parameters]

total	Specifies the number of packet data arrays.
data	Specifies a pointer to the packet data array to be transmitted.

[Return value]

None

[Remarks]

This function is supported only for devices that include a USB function.

Function for transmitting signals from HOST using USB function

```
void        UpSetUSBSig(sig)
unsigned    char    sig;                        /* Transmit signal ID */
```

[Function]

This function uses the USB function to transmit a signal from the HOST.

[Parameter]

sig	Specifies the transmit signal ID. 0: USBreset 1: Resume

[Return value]

None

[Remarks]

This function is supported only for devices that include a USB function.

Function for receiving signals from Function using USB function

```
void        UpGetUSBSig(sig)
unsigned    char     *sig;                /* Receive signal ID */
```

[Function]

This function uses the USB function to receive a signal from Function.

[Parameter]

sig	Specifies the receive signal ID. 0: USBreset 1: Resume

[Return value]

None

[Remarks]

This function is supported only for devices that include a USB function.

CHAPTER 5 OPERATIONS DURING CPU RESET

This chapter describes the operations of customized parts when a CPU reset is triggered by the simulator debugger.

5.1 Parts Customized via Parts Window

The parts for functions that are specified for customization via the Parts window are listed below.

Table 5-1. Parts Customized via Parts Window During CPU Reset

Part Name	Status
Push button	All are set to inactive mode.
Toggle button	All are set to inactive mode.
Group select button	All are set to non-pressed mode.
LED	All are set to inactive mode.
LED set per port	All are set to inactive mode.
Matrix LED	All are set to OFF mode.
DC motor	All are set to inactive mode and total active time is set to 0.
Stepping motor	All are set to inactive mode and the number of positive revolutions, the number of negative revolutions, and rotation angle are all set to 0.
Vertical scroll bar analog input	Input value is set to 0 and scroll bar's scroll button is set to the bottom edge.
Serial pin data input	Returns to start of data.

5.2 Parts Customized via User Window

When a CPU reset has been triggered by the simulator debugger, if the function name has already been reported by the reset function name notification function UpResetFuncName(), the user window's reset processing function is performed.

CHAPTER 6 PROGRAMMING EXAMPLES

This chapter presents some examples of customized parts.

Among the sources cited below:

<1> refers to the target program.
Programs are compiled and linked using the CC78K Series to create load module files (xxxx.lmf).

<2> and subsequent sources refer to files that are required when creating customized parts.
This manual specifies that Visual C++ is used to create dynamic link libraries (xxxx.DLL).
When compiling, be sure to specify the /Zp1 option. (/Zp1 option: Sets single-byte alignment of structure members)
Select [C/C++] from [Setting] on the [Project] menu bar and set “1 byte” for the “Structure alignment” in “Category” → “Code Generation”.

Remark If using the SM78K Series V2.30 or a later version, be sure to create 32-bit dynamic link libraries.

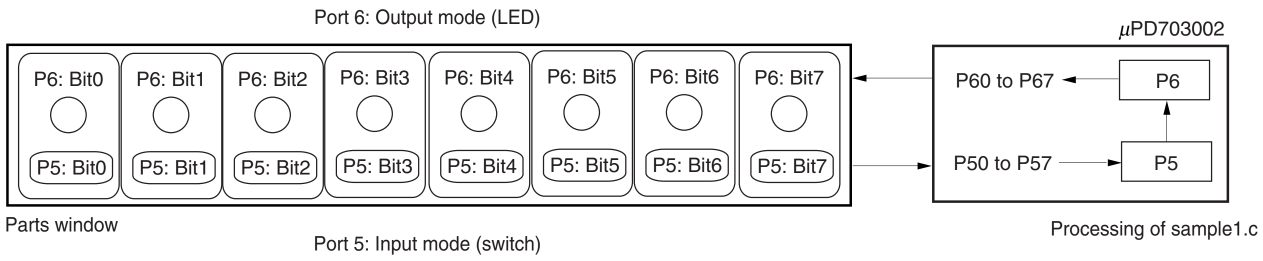
6.1 Example of Parts Customized via Parts Window

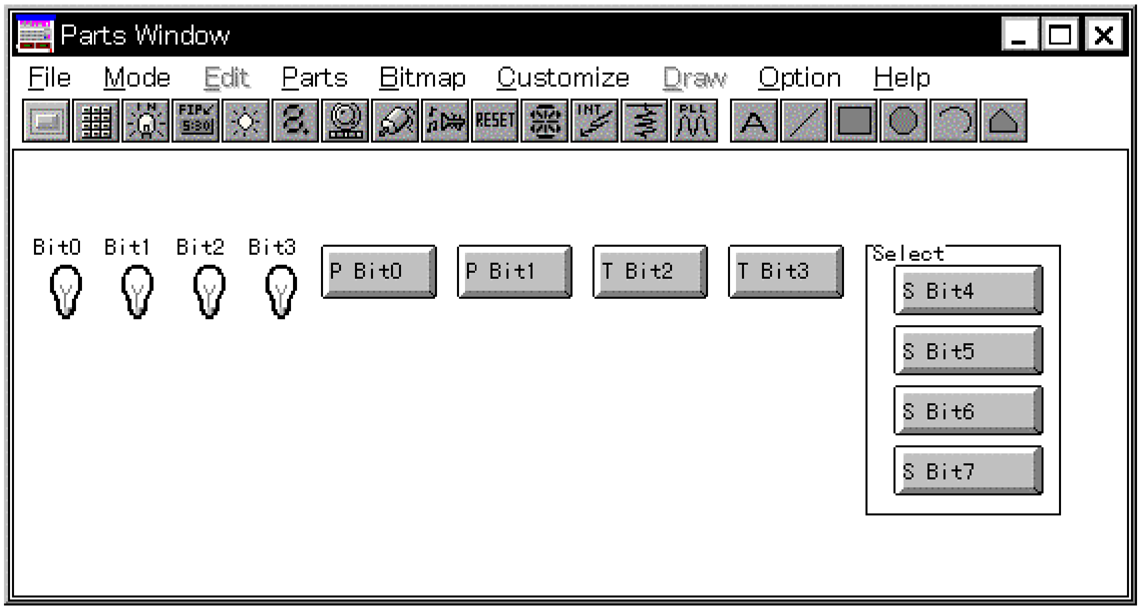
6.1.1 Description of samples

The items displayed in the Parts window include eight LEDs and eight switches, of which two (P50 and P51) are push buttons, two (P52 and P53) are toggle buttons, and four (P54, P55, P56, and P57) are select buttons. When a switch is set ON or OFF, its corresponding LED is also set ON or OFF.

An example is shown below.

Figure 6-1. Example of Parts Customized via Parts Window





6.1.2 Source examples

<1> Target program

(1/1) SAMPLE1.C

```
#pragma sfr
void main()
{
       MM = 0xB0;
       PM5 = 0xFF;
       PM6 = 0;
       P6 = 0;

       while(1)
       {
               P6 = P5;
       }
}
```

<2> Custom part source file UPsw00.c

(1/2) UPsw00.c

```
/*
 *     User Open I/F  Sample Program   (UPsw00.c)
 *
 *     P50 (I) : Switch 0      P60 (O) : LED 0
 *     P51 (I) : Switch 1      P61 (O) : LED 1
 *     P52 (I) : Switch 2      P62 (O) : LED 2
 *     P53 (I) : Switch 3      P63 (O) : LED 3
 *     P54 (I) : Switch 4      P64 (O) : LED 4
 *     P55 (I) : Switch 5      P65 (O) : LED 5
 *     P56 (I) : Switch 6      P66 (O) : LED 6
 *     P57 (I) : Switch 7      P67 (O) : LED 7
 */

#include       <Windows.h>
#include       <string.h>

typedef unsigned char  UCHAR;
typedef unsigned short USHORT;
typedef unsigned long  ULONG;


  #include     "uparts32.h"

BOOL APIENTRY DllMain(HANDLE, DWORD, LPVOID);

void FAR PASCAL       UParts_sw00(void);


/************************************************************************/
/*    DLL Main                                                          */
/************************************************************************/
BOOL APIENTRY DllMain(HANDLE hModele, DWORD ul_reason_for_call, LPVOID lpReserved)
  {
      return(TRUE);
  }
```

(2/2) UPsw00.c

```
/***********************************************************************/
/*    UParts_sw00(void)                                                */
/***********************************************************************/
void FAR PASCAL
UParts_sw00(void)
{
    static char *pin[4]  = { "P54", "P55", "P56", "P57" };
    static char *name[4] = { "S Bit4", "S Bit5", "S Bit6", "S Bit7" };

    UpSetPBtmtime("3.0");
    UpLed("P60", HIGH, "Bit0", 1);
    UpLed("P61", HIGH, "Bit1", 1);
    UpLed("P62", HIGH, "Bit2", 1);
    UpLed("P63", HIGH, "Bit3", 1);
    UpLed("P64", HIGH, "Bit4", 1);
    UpLed("P65", HIGH, "Bit5", 1);
    UpLed("P66", HIGH, "Bit6", 1);
    UpLed("P67", HIGH, "Bit7", 1);
    UpPushBtm("P50", HIGH, "P Bit0");
    UpPushBtm("P51", HIGH, "P Bit1");
    UpTglBtm("P52", HIGH, "T Bit2");
    UpTglBtm("P53", HIGH, "T Bit3");
    UpSelectBtm("Select", pin, 4, HIGH, name);
}

/* UPsw00.c */
```

<3> Definition file UPsw00.def

(1/1) UPsw00.def

```
LIBRARY         UPSW00

DESCRIPTION     'User Open I/F Panel sw00'

CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD         SINGLE

HEAPSIZE        3072

EXPORTS
        UParts_sw00                     @2
```

<4> Make file UPsw00.mak

(1/4) UPsw00.mak

```
# Microsoft Developer Studio Generated NMAKE File, Based on UOport.dsp
!IF "$(CFG)" == ""
CFG=UOport - Win32 Release
!MESSAGE Configuration not specified. Set default upsw00 - Win32 Debug.
!ENDIF
!IF "$(CFG)" != "UOport - Win32 Release" && "$(CFG)" != "upsw00 - Win32 Debug"
!MESSAGE Specified build mode “$ (CFG)” is not correct.
!MESSAGE Configuration can be specified during execution of NMAKE.
!MESSAGE Defines command-line macro setting. Example:
!MESSAGE
!MESSAGE NMAKE /f "upsw00.mak" CFG="upsw00 - Win32 Debug"
!MESSAGE
!MESSAGE Selectable build modes:
!MESSAGE
!MESSAGE “upsw00 - Win32 Release” (for “Win32 (x86) Dynamic-Link Library”)
!MESSAGE “upsw00 - Win32 Debug” (for “Win32 (x86) Dynamic-Link Library”)
!MESSAGE
!ERROR Invalid configuration was specified.
!ENDIF
!IF "$(OS)" == "Windows_NT"
NULL=
!ELSE
NULL=nul
!ENDIF
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe
!IF  "$(CFG)" == "upsw00 - Win32 Release"
OUTDIR=.\Release
INTDIR=.\Release
# Begin Custom Macros
OutDir=.\Release
# End Custom Macros
ALL : "$(OUTDIR)\upsw00.dll"
```

(2/4) UPsw00.mak

```
CLEAN :
       -@erase "$(INTDIR)\Upsw00.obj"
       -@erase "$(INTDIR)\vc60.idb"
       -@erase "$(OUTDIR)\upsw00.dll"
       -@erase "$(OUTDIR)\upsw00.exp"
       -@erase "$(OUTDIR)\upsw00.lib"
"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"
CPP_PROJ=/nologo /Zp1 /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_MBCS" /D "_USRDLL" /D "UPSW00_EXPORTS" /Fp"$(INTDIR)\upsw00.pch" /YX
/Fo"$(INTDIR)\\" /Fd"$(INTDIR)\\" /FD /c
MTL_PROJ=/nologo /D "NDEBUG" /mktyplib203 /win32
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$(OUTDIR)\upsw00.bsc"
BSC32_SBRS= \

LINK32=link.exe
LINK32_FLAGS=kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /incremental:no /pdb:"$(OUTDIR)\upsw00.pdb" /machine:I386
/def:".\Upsw00.def" /out:"$(OUTDIR)\upsw00.dll" /implib:"$(OUTDIR)\upsw00.lib"
DEF_FILE= \
       ".\Upsw00.def"
LINK32_OBJS= \
       "$(INTDIR)\Upsw00.obj" \
       ".\sik032.lib"
"$(OUTDIR)\upsw00.dll" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<
!ELSEIF  "$(CFG)" == "upsw00 - Win32 Debug"
OUTDIR=.\Debug
INTDIR=.\Debug
# Begin Custom Macros
OutDir=.\Debug
# End Custom Macros
ALL : "$(OUTDIR)\upsw00.dll"
```

(3/4) UPsw00.mak

```
CLEAN :
       -@erase "$(INTDIR)\Upsw00.obj"
       -@erase "$(INTDIR)\vc60.idb"
       -@erase "$(INTDIR)\vc60.pdb"
       -@erase "$(OUTDIR)\upsw00.dll"
       -@erase "$(OUTDIR)\upsw00.exp"
       -@erase "$(OUTDIR)\upsw00.ilk"
       -@erase "$(OUTDIR)\upsw00.lib"
       -@erase "$(OUTDIR)\upsw00.pdb"
"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"
CPP_PROJ=/nologo /Gz /Zp1 /MTd /W3 /Gm /GX /Zi /Od /D "WIN32" /D "_DEBUG" /D
"_WINDOWS" /D "_MBCS" /D "_USRDLL" /D "UPSW00_EXPORTS" /Fp"$(INTDIR)\upsw00.pch" /YX
/Fo"$(INTDIR)\\" /Fd"$(INTDIR)\\" /FD /c
MTL_PROJ=/nologo /D "_DEBUG" /mktyplib203 /win32
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$(OUTDIR)\upsw00.bsc"
BSC32_SBRS= \

LINK32=link.exe
LINK32_FLAGS=kernel32.lib    user32.lib    gdi32.lib    winspool.lib    comdlg32.lib
advapi32.lib  shell32.lib  ole32.lib  oleaut32.lib  uuid.lib  odbc32.lib  odbccp32.lib
/nologo   /dll   /incremental:yes   /pdb:"$(OUTDIR)\upsw00.pdb"   /debug   /machine:I386
/def:".\Upsw00.def"    /out:"$(OUTDIR)\upsw00.dll"    /implib:"$(OUTDIR)\upsw00.lib"
/pdbtype:sept
DEF_FILE= \
       ".\Upsw00.def"
LINK32_OBJS= \
       "$(INTDIR)\Upsw00.obj" \
       ".\sik032.lib"
"$(OUTDIR)\upsw00.dll" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<
!ENDIF
.c{$(INTDIR)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
```

(4/4) UPsw00.mak

```
.cpp{$(INTDIR)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
.cxx{$(INTDIR)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
.c{$(INTDIR)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
.cpp{$(INTDIR)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
.cxx{$(INTDIR)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
!IF "$(NO_EXTERNAL_DEPS)" != "1"
!IF EXISTS("upsw00.dep")
!INCLUDE "upsw00.dep"
!ELSE
!MESSAGE Warning: cannot find "upsw00.dep"
!ENDIF
!ENDIF
!IF "$(CFG)" == "upsw00 - Win32 Release" || "$(CFG)" == "upsw00 - Win32 Debug"
SOURCE=.\Upsw00.c
"$(INTDIR)\Upsw00.obj" : $(SOURCE) "$(INTDIR)"
!ENDIF
```

6.2 Example of Parts Customized via User Window

6.2.1 Description of samples

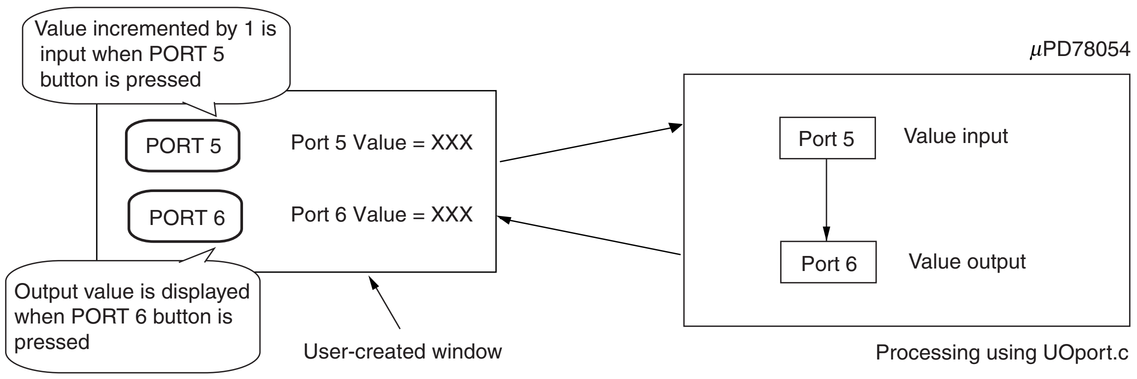
The items displayed in the user custom window include a part that sets a value to port 5 and a part that captures values from port 6.

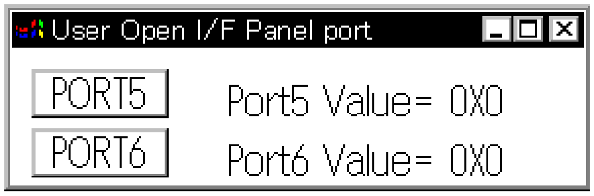
Pressing the PORT 5 button in the user custom window causes the value input to port 5 to be incremented by 1. When "Port 5 Value = 0x7f", the value is cleared to 0.

This input value from port 5 is output to port 6. The display of "Port 6 Value = XXXX" is updated to the current value when the PORT 6 button in the user custom window is clicked.

An example is shown below.

Figure 6-2. Example of Parts Customized via User Window


Value incremented by 1 is input when PORT 5 button is pressed
PORT 5
Port 5 Value = XXX
PORT 6
Port 6 Value = XXX
Output value is displayed when PORT 6 button is pressed
User-created window
μPD78054
Port 5
Value input
Port 6
Value output
Processing using UOport.c



6.2.2 Source examples

<1> Target program

(1/1) SAMPLE2.C

```
#pragma sfr

void main()
{
        MM = 0xB0;
        PM5 = 0xFF;
        PM6 = 0;
        P6 = 0;

        while(1)
        {
                P6 = P5;
        }
}
```

<2> Custom part source file UOport.c

(1/7) UOport.c

```
/*
 *     User Open I/F  Sample Program   (UOport.c)
 */

#include       <stdio.h>
#include       <stdlib.h>
#include       <string.h>

typedef unsigned char  UCHAR;
typedef unsigned short USHORT;
typedef unsigned long  ULONG;

#include       <Windows.h>
#include       "uparts32.h"

#define IDM_PAST       0x1111
#define IDM_NEWWIN     0x1112
#define BTN_WIDTH      70
#define BTN_HIGHT      25
#define IDD_PIN_BUTTON 0x10

void FAR PASCAL        UParts_port(void);
LONG FAR PASCAL        UParts_portWndProc(HWND, unsigned, WPARAM, LPARAM);
void FAR PASCAL        UParts_portCall(ULONG);
void FAR PASCAL        UParts_portReset(void);
void FAR PASCAL        UParts_portLoadProj(char FAR *);
void FAR PASCAL        UParts_portSaveProj(char FAR *);

/* Window point */
#define        UParts_portWIDTH        300
#define        UParts_portHEIGHT       100

/* Title Strings */
#define        STR_UP_TITLE     "User Open I/F Panel port"

/* Window Class Name */
const BYTE      cnUParts_port[] = "UParts_portWin";
```

(2/7) UOport.c

```
HANDLE hInst;
HWND   hUParts_portWnd;
HWND   btm_hwnd[2];
char   *strbuf[2] = {"PORT5","PORT6"};
char   port5val = 0;
char   port6val = 0;
char   UParts_Veiw_str[7];
char   UParts_Rect_str[23];

/*************************************************************************/
/*     DLL Main                                                        */
/*************************************************************************/

BOOL APIENTRY DllMain(HANDLE hModele, DWORD ul_reason_for_call, LPVOID lpReserved)
{
    WNDCLASS   wndclass;

    switch(ul_reason_for_call){
       case DLL_PROCESS_ATTACH:
            hInst = hModele;
            wndclass.lpszClassName = (LPSTR)cnUParts_port;
            wndclass.hInstance     = hInst;
            wndclass.lpfnWndProc   = (WNDPROC)UParts_portWndProc;
            wndclass.hCursor      = NULL;
            wndclass.hIcon        = NULL;
            wndclass.lpszMenuName  = NULL;
            wndclass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
            wndclass.style        = CS_HREDRAW | CS_VREDRAW;
            wndclass.cbClsExtra           = 0;
            wndclass.cbWndExtra    = DLGWINDOWEXTRA;

            RegisterClass(&wndclass);

            break;
       case DLL_THREAD_ATTACH:
           break;
       case DLL_THREAD_DETACH:
            break;
       case DLL_PROCESS_DETACH:
           break;
```

(3/7) UOport.c

```
}

    return(TRUE);
}
/************************************************************************/
/*     UParts_port(void)                                                */
/************************************************************************/
void FAR PASCAL UParts_port(void)
{
    if(!hUParts_portWnd){
       hUParts_portWnd = CreateWindow((LPSTR)cnUParts_port,/* Class name */
       STR_UP_TITLE,                                   /* Title.       */
       WS_OVERLAPPEDWINDOW | WS_BORDER | WS_VISIBLE,   /* Style bits.  */
       CW_USEDEFAULT,                                  /* x - default. */
       CW_USEDEFAULT,                                  /* y - default. */
       UParts_portWIDTH,                               /* cx - default.*/
       UParts_portHEIGHT,                              /* cy - default.*/
       NULL,                                           /* No parent.   */
       NULL,                                           /* Class memu.  */
       hInst,                                          /* Creator      */
       NULL);                                          /* Params.      */
    }
    if(hUParts_portWnd){
       UpSetUserWnd(hUParts_portWnd);
       ShowWindow(hUParts_portWnd, SW_SHOW);
       UpCallFuncName("UParts_portCall");
       UpResetFuncName("UParts_portReset");
       UpLoadProjName("UParts_portLoadProj");
       UpSaveProjName("UParts_portSaveProj");
       UpInitPort("P5", 0xFF);
    }
    return;
}
```

(4/7) UOport.c

```
/***********************************************************************/
/*     UParts_portWndProc(HWND, unsigned WPARAM, LPARAM)                   */
/***********************************************************************/
LONG  PASCAL  FAR  UParts_portWndProc(HWND  hWnd,  unsigned  iMessage,  WPARAM  wParam,
LPARAM lParam)
{
    HDC                hDC;
    PAINTSTRUCT        ps;
    RECT       wRect;
    int                i;
    char       strval[20];
    long       wx,wy;
    switch(iMessage){
    case WM_CREATE:
       for(i = 0; i < 2; i++){
               btm_hwnd[i] = CreateWindow((LPSTR)"button",strbuf[i],
                       WS_CHILD|BS_PUSHBUTTON|WS_VISIBLE|WS_TABSTOP,
                        10,10+30*i,
                        BTN_WIDTH,BTN_HIGHT,
                        hWnd,(HMENU)(IDD_PIN_BUTTON+i),hInst,NULL);
       }
       return(FALSE);
    case WM_COMMAND:
       switch(wParam){
           case IDD_PIN_BUTTON:
                if(port5val < 0x7f)
                     port5val++;
                else
                    port5val = 0;
                        UpSetPort("P5", port5val, 0);
                InvalidateRect(hWnd, NULL, TRUE);
                UpdateWindow(hWnd);
                break;
          case IDD_PIN_BUTTON + 1:
                UpGetPort("P6", &port6val);
                InvalidateRect(hWnd, NULL, TRUE);
                UpdateWindow(hWnd);
                break;
       }
```

(5/7) UOport.c

```
return(FALSE);
   case WM_PAINT:
       hDC = BeginPaint(hWnd, &ps);
       wsprintf(strval, "Port5 Value=%#2X\0", port5val);
       TextOut(hDC, BTN_WIDTH + 40, 15, strval, lstrlen(strval));
       wsprintf(strval, "Port6 Value=%#2X\0", port6val);
       TextOut(hDC, BTN_WIDTH + 40, 45, strval, lstrlen(strval));
       EndPaint(hWnd, &ps);
       return(FALSE);
   case WM_SYSCOLORCHANGE:
       InvalidateRect(hWnd, NULL, TRUE);
       break;
   case WM_MOVE:
       GetWindowRect(hWnd, &wRect);
       wx = wRect.right - wRect.left;
       wy = wRect.bottom - wRect.top;
       if((wx != 36) && (wy != 36)) {
               wsprintf(UParts_Rect_str, "%d, %d, %d, %d",
                               wRect.left, wRect.top, wx, wy);
       }
       InvalidateRect(hWnd, NULL, TRUE);
       break;
   case WM_SIZE:
       if(wParam == SIZEICONIC) {
               lstrcpy(UParts_Veiw_str, "Icon");
       } else {
               GetWindowRect(hWnd, &wRect);
               lstrcpy(UParts_Veiw_str, "Normal");
               wsprintf(UParts_Rect_str, "%d, %d, %d, %d",
                               wRect.left,
                               wRect.top,
                               wRect.right - wRect.left,
                               wRect.bottom - wRect.top);
       }
       break;
   case WM_DESTROY:
       UpCloseUserWnd(hWnd);
   default:
       return DefWindowProc(hWnd, iMessage, wParam, lParam);
   }
```

(6/7) UOport.c

```
   return 0L;
}
/**********************************************************************/
/*     UParts_portCall(ULONG)                                         */
/**********************************************************************/
void FAR PASCAL
UParts_portCall(ULONG time)
{
       return;
}
/**********************************************************************/
/*     UParts_portReset(void)                                         */
/**********************************************************************/
void FAR PASCAL
UParts_portReset(void)
{
    port5val = 0;
    port6val = 0;
    InvalidateRect(hUParts_portWnd, NULL, TRUE);
}
/**********************************************************************/
/*     UParts_portLoadProj(char FAR *)                                */
/**********************************************************************/
void FAR PASCAL
UParts_portLoadProj(char FAR *fname)
{
    char       *next;
    WORD       x, y, wx, wy;
    GetPrivateProfileString("UOport", "Window",
                              "Hide", UParts_Veiw_str, 7, fname);
    if(!lstrcmp(UParts_Veiw_str, "Icon")) {    /* "Icon" mode */
       ShowWindow(hUParts_portWnd, SW_SHOWMINNOACTIVE);
    } else {                                   /* "Normal" mode */
       GetPrivateProfileString("UOport", "Geometry",
                              "0, 0, 0, 0", UParts_Rect_str, 23, fname);
       if(lstrcmp(UParts_Rect_str, "0, 0, 0, 0")) {
           next = strtok(UParts_Rect_str, ",");
           x = (WORD)strtoul(next, NULL, 10);
           next = strtok(NULL, ",");
           y = (WORD)strtoul(next, NULL, 10);
           next = strtok(NULL, ",");
```

(7/7) UOport.c

```
            wx = (WORD)strtoul(next, NULL, 10);
            next = strtok(NULL, "");
            wy = (WORD)strtoul(next, NULL, 10);
            MoveWindow(hUParts_portWnd, x, y, wx, wy, TRUE);
        }
        ShowWindow(hUParts_portWnd, SW_SHOWNORMAL);
    }
}
/*************************************************************************/
/*     UParts_portSaveProj(char FAR *)                          */
/*************************************************************************/
void FAR PASCAL
UParts_portSaveProj(char FAR *fname)
{
    WritePrivateProfileString("UOport", "Window",   UParts_Veiw_str, fname);
    WritePrivateProfileString("UOport", "Geometry", UParts_Rect_str, fname);
}
/* UOport.c */
```

<3> Definition file UOport.def

(1/1) UOport.def

```
LIBRARY                UOPORT

DESCRIPTION    'User Open I/F Panel port'

HEAPSIZE       4096

EXPORTS
               UParts_port
               UParts_portWndProc
               UParts_portCall
               UParts_portReset
               UParts_portLoadProj
               UParts_portSaveProj
```

<4> Make file UOport.mak

(1/4) UOport.mak

```
# Microsoft Developer Studio Generated NMAKE File, Based on uoadda00.dsp
!IF "$(CFG)" == ""
CFG=uoadda00 - Win32 Debug
!MESSAGE Configuration not specified.  Set default uoadda00 – Win32 Debug.
!ENDIF
!IF "$(CFG)" != "uoadda00 - Win32 Release" && "$(CFG)" != "uoadda00 - Win32 Debug"
!MESSAGE Specified build mode "$(CFG)" is not correct.
!MESSAGE Configuration can be specified during execution of NMAKE.
!MESSAGE Defines command-line macro setting.  Example:
!MESSAGE
!MESSAGE NMAKE /f "uoadda00.mak" CFG="uoadda00 - Win32 Debug"
!MESSAGE
!MESSAGE Selectable build modes:
!MESSAGE
!MESSAGE "uoadda00 – Win32 Release" (for "Win32 (x86) Dynamic-Link Library")
!MESSAGE "uoadda00 – Win32 Debug" (for "Win32 (x86) Dynamic-Link Library")
!MESSAGE
!ERROR   Invalid configuration was specified.
!ENDIF
!IF "$(OS)" == "Windows_NT"
NULL=
!ELSE
NULL=nul
!ENDIF
CPP=cl.exe
MTL=midl.exe
RSC=rc.exe
!IF  "$(CFG)" == "uoadda00 - Win32 Release"
OutDir=.\Release
INTDIR=.\Release
# Begin Custom Macros
OUTDIR=.\Release
# End Custom Macros
ALL : "$(OUTDIR)\UOport.dll"
```

(2/4) UOport.mak

```
CLEAN :
       -@erase "$(INTDIR)\Uoport.obj"
       -@erase "$(INTDIR)\vc60.idb"
       -@erase "$(OUTDIR)\UOport.dll"
       -@erase "$(OUTDIR)\UOport.exp"
       -@erase "$(OUTDIR)\UOport.lib"
"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"
CPP_PROJ=/nologo /Zp1 /MT /W3 /GX /O2 /D "WIN32" /D "NDEBUG" /D "_WINDOWS" /D
"_MBCS" /D "_USRDLL" /D "UOPORT_EXPORTS" /Fp"$(INTDIR)\UOport.pch" /YX
/Fo"$(INTDIR)\\" /Fd"$(INTDIR)\\" /FD /c
MTL_PROJ=/nologo /D "NDEBUG" /mktyplib203 /win32
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$(OUTDIR)\UOport.bsc"
BSC32_SBRS= \

LINK32=link.exe
LINK32_FLAGS=kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo /dll /incremental:no /pdb:"$(OUTDIR)\UOport.pdb" /machine:I386
/def:".\Uoport.def" /out:"$(OUTDIR)\UOport.dll" /implib:"$(OUTDIR)\UOport.lib"
DEF_FILE= \
       ".\Uoport.def"
LINK32_OBJS= \
       "$(INTDIR)\Uoport.obj" \
       ".\sik032.lib"
"$(OUTDIR)\UOport.dll" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<
!ELSEIF  "$(CFG)" == "UOport - Win32 Debug"
OUTDIR=.\Debug
INTDIR=.\Debug
# Begin Custom Macros
OutDir=.\Debug
# End Custom Macros
ALL : "$(OUTDIR)\UOport.dll"
```

(3/4) UOport.mak

```
CLEAN :
       -@erase "$(INTDIR)\Uoport.obj"
       -@erase "$(INTDIR)\vc60.idb"
       -@erase "$(INTDIR)\vc60.pdb"
       -@erase "$(OUTDIR)\UOport.dll"
       -@erase "$(OUTDIR)\UOport.exp"
       -@erase "$(OUTDIR)\UOport.ilk"
       -@erase "$(OUTDIR)\UOport.lib"
       -@erase "$(OUTDIR)\UOport.pdb"
"$(OUTDIR)" :
    if not exist "$(OUTDIR)/$(NULL)" mkdir "$(OUTDIR)"
CPP_PROJ=/nologo /Zp1 /MTd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS"
/D  "_MBCS"  /D  "_USRDLL"  /D  "UOPORT_EXPORTS"  /Fp"$(INTDIR)\UOport.pch"  /YX
/Fo"$(INTDIR)\\" /Fd"$(INTDIR)\\" /FD /GZ /c
MTL_PROJ=/nologo /D "_DEBUG" /mktyplib203 /win32
BSC32=bscmake.exe
BSC32_FLAGS=/nologo /o"$(OUTDIR)\UOport.bsc"
BSC32_SBRS= \

LINK32=link.exe
LINK32_FLAGS=kernel32.lib   user32.lib   gdi32.lib   winspool.lib   comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
/nologo  /dll  /incremental:yes  /pdb:"$(OUTDIR)\UOport.pdb"  /debug  /machine:I386
/def:".\Uoport.def"   /out:"$(OUTDIR)\UOport.dll"   /implib:"$(OUTDIR)\UOport.lib"
/pdbtype:sept
DEF_FILE= \
       ".\Uoport.def"
LINK32_OBJS= \
       "$(INTDIR)\Uoport.obj" \
       ".\sik032.lib"
"$(OUTDIR)\UOport.dll" : "$(OUTDIR)" $(DEF_FILE) $(LINK32_OBJS)
    $(LINK32) @<<
  $(LINK32_FLAGS) $(LINK32_OBJS)
<<
!ENDIF
.c{$(INTDIR)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
```

(4/4) UOport.mak

```
.cpp{$(INTDIR)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
.cxx{$(INTDIR)}.obj::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
.c{$(INTDIR)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
.cpp{$(INTDIR)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
.cxx{$(INTDIR)}.sbr::
   $(CPP) @<<
   $(CPP_PROJ) $<
<<
!IF "$(NO_EXTERNAL_DEPS)" != "1"
!IF EXISTS("UOport.dep")
!INCLUDE "UOport.dep"
!ELSE
!MESSAGE Warning: cannot find "UOport.dep"
!ENDIF
!ENDIF
!IF "$(CFG)" == "UOport - Win32 Release" || "$(CFG)" == "UOport - Win32 Debug"
SOURCE=.\Uoport.c
"$(INTDIR)\Uoport.obj" : $(SOURCE) "$(INTDIR)"
!ENDIF
```

APPENDIX A ERROR MESSAGES

A.1 Error Processing

(a) If the specified pin name is not among the products that can be simulated, the Error Message dialog box appears to report an error message.

(b) If the read DLL file is a combination of user panel custom functions and Parts custom functions, a dialog box appears with a warning message when the first function to be read does not belong to the DLL file in accordance with the DLL file name.

(c) If an error occurs when a user-created custom DLL is read, the part that caused the error is not created.

(d) If an error or warning occurs even once for the user panel custom functions UpGetPin(), UpGetPort(), UpGetMem(), UpClrMtrAcClk(), or UpGetStpingMtr(), error values may be returned or the function may not operate correctly when subsequently used. Therefore, if an error or warning occurs, revise the source code, create the DLL file again, and reload to avoid such problems.

A.2 Error and Warning Messages

Error messages and warning messages that may occur during execution of a function are listed below. The abbreviated function names listed below are used to refer to the function names for which the error occurred.

Stepper motor functions	UpStpingMtr(), UpSetStpingMtr(), UpGetStpingMtr()
LED picture setup function	UpSetLedPic()
LED functions	UpLed(), UpPortLed()
Matrix LED function	UpMtxLed()
Serial pin data input function	UpSerial_data()
Port value setup/capture functions	UpPortLed(), UpGetPort(), UpSetPort()
Hold time setup function	UpSetPBtmtime()
Vertical scroll bar analog input function	UpScaleInterAD()
Reference voltage value setup function	UpSetAVref()
Function name notification functions	UpCallFuncName(), UpLoadProjName(), UpSaveProjName(), UpResetFuncName()
Bitmap setup functions	UpSetBtmBmp(), UpSetLedBmp(), UpSetMtrBmp()
Button functions	UpPushBtm(), UpTglBmp(), UpSelectBtm()
Register-related functions	UpGetReg(), UpSetReg ()

Table A-1. Error Messages (1/2)

No.	Window Name	Message	Cause	User Action
E1	All functions that include a pin name parameter	Specified pin does not exist.	Specified pin does not exist.	Specify a pin name that exists in the target device.
E2	All functions that include a pin name parameter	Pin name was specified using double-byte characters.	Specified pin name is entered using double-byte characters.	Use only single-byte characters to specify pin names.
E3	All functions that include an active high/low parameter	Active high/low setting is neither HIGH nor LOW.	Specified active high/low setting is neither HIGH nor LOW.	Specify either HIGH or LOW for the active high/low setting.
E4	Stepper motor functions that include a number-of-channels parameter	The number of channels is neither 4 nor 8.	Specified number of channels is neither 4 nor 8.	Specify either 4 or 8 as the number of channels (according to the number of pins).
E5	Stepper motor functions that include an excitation parameter	Excitation value is neither 0 nor 1.	Specified excitation value is neither 0 nor 1.	Specify either 0 or 1 as the excitation value (according to the excitation method).
E6	Stepper motor functions that include a minimum step angle parameter	Minimum step angle is not a fraction of 360.	Specified minimum step angle is not a fraction of 360.	Specify an integer that is a fraction of 360 as the minimum step angle.
E7	LED picture setup function	Picture type is neither PIC_RECT nor PIC_ELL.	Specified picture type parameter is neither PIC_RECT nor PIC_ELL.	Specify either PIC_RECT or PIC_ELL as the picture type.
E8	LED picture setup function	Color type is neither PIC_RED nor PIC_YELLOW nor PIC_GREEN.	Specified color type parameter is neither PIC_RED nor PIC_YELLOW nor PIC_GREEN.	Specify PIC_RED, PIC_YELLOW, or PIC_GREEN as the color type.
E9	LED functions that include a picture type parameter	Displayed picture type specification is neither 0 nor 1.	Specified picture type value is neither 0 nor 1.	Specify either 0 or 1 as the picture type parameter.
E10	Serial pin data input function	The first bit specification for serial input is neither MSB (1) nor LSB (0).	Value specified as the first bit parameter is neither 0 nor 1.	Specify either 0 or 1 as the first bit parameter.
E11	Matrix LED function	Output 1 active high/low setting is neither HIGH nor LOW.	Specified value for output 1 active high/low parameter is neither HIGH nor LOW.	Specify either HIGH or LOW as the output 1 active high/low parameter.
E12	Matrix LED function	Output 2 active high/low setting is neither HIGH nor LOW.	Specified value for output 2 active high/low parameter is neither HIGH nor LOW.	Specify either HIGH or LOW as the output 2 active high/low parameter.
E13	Port value setup/capture functions	Port name was specified using double-byte characters.	Specified port name parameter was entered using double-byte characters.	Use only single-byte characters to specify port name parameter.
E14	Port value setup/capture functions	Specified port does not exist.	Specified parameter for port name does not exist.	Specify a port name that exists in the target device.

Table A-1. Error Messages (2/2)

No.	Window Name	Message	Cause	User Action
E15	Hold time setup function	Hold time is invalid.	Specified hold time is out of range or is not a number.	Specify a hold time within the range of 0.001 to 999 ms.
E16	Vertical scroll bar analog input function	Specified pin is not an analog input pin.	Non-analog pin name was specified.	Specify an analog pin.
E17	Reference voltage value setup function	AV_{REF} is not within the operating power supply voltage range.	AV_{REF} is not within the operating power supply voltage range.	Set a value that is within the operating power supply voltage range.
E18	All functions that include a pointer type parameter	Parameter is a NULL pointer.	An invalid parameter was specified for the function.	Specify the correct pointer.
E19	All functions described in section 4.1	Function XXXX cannot be specified in UOxxx.dll.	This function was specified in UOxxx.dll.	Use UPxxx.dll.
E20	All functions described in section 4.2	Function XXXX cannot be specified in UPxxx.dll.	This function was specified in UPxxx.dll.	Use UOxxx.dll.
E21	Functions requiring advance notification function	No advance notification of function XXXX	Required notification function was not called.	Call the required notification function in advance.
E22	Register-related functions	Type of control register is neither REG_PC, nor REG-PSW, nor REG-SP.	Register other than a control register is specified.	Do not specify register other than control register
E23	Bitmap setup functions	Specified bitmap file is invalid.	Access to specified file or area allocation failed.	Specify a correct file, or remove other applications to allocate memory.
E24	Button functions	Specified pin has already been specified.	Attempted to specify previously specified pin.	Do not specify a previously specified pin.
E25	All functions	Area cannot be allocated.	Memory cannot be allocated.	Remove other applications to allocate memory.

Table A-2. Warning Messages

No.	Window Name	Message	Cause	User Action
W1	Reference voltage value setup function	No AV_{REF} setting. Is 5.0 V (default setting) OK?	" " was specified as the reference voltage value character string.	Specify a character string as the values to be set to the reference voltage value character string.
W2	Hold time setup function	No hold time setting. Is 0.5 ms (default setting) OK?	" " was specified as the hold time character string.	Specify a character string as the values to be set to the hold time character string.

NEC

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: +1-800-729-9288
+1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Market Communication Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6462-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: +82-2-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: +886-2-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title:

Document number: Page number:

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❏	❏	❏	❏
Technical Accuracy	❏	❏	❏	❏
Organization	❏	❏	❏	❏

CS 01.11