REJ10J1822-0101

Everywhere you imagine.

Stub Generator

V.1.00

User's Manual

Renesas Microcomputer Development Environment System

User's Manual

Rev.1.01
Revision Date: Mar. 17, 2008

Renesas Technology
www.renesas.com

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.

Preface

This manual describes how to use the stub generator. Read this manual and understand it well before using the stub generator.

Notes on Descriptions

RPCGEN	Abbreviation for the stub generator
Prefix	Prefix 0x indicates a hexadecimal number. Numbers with no prefix are decimal.
\	'\' is the directory delimiter.
[Menu -> Menu item]	'->' leads to the menu item (e.g. File -> Save).
$(xxxx)	Custom placeholder in the High-performance Embedded Workshop

Trademarks

All trademarks and registered trademarks belong to their respective owners.

1. TRON is an acronym formed from "The Real Time Operating system Nucleus". ITRON is short for "Industrial TRON" and μITRON is short for "Micro Industrial TRON".

   TRON, ITRON, and μITRON are the names of computer specifications and do not indicate specific products or groups of products.

   The μITRON4.0 specification is an open realtime-kernel specification defined by the TRON Association. The μITRON4.0 specification can be downloaded from the TRON Association homepage (http://www.assoc.tron.org).

   The copyright of the μITRON specification belongs to the TRON Association.
2. Microsoft® and Windows® are registered trademarks of Microsoft Corporation in the United States and/or other countries. The formal name of Windows® is Microsoft Windows Operating System.
3. All other product names are trademarks or registered trademarks of the respective holders.

Renesas Technology Homepage

Various support information are available on the following Renesas Technology homepage:

http://www.renesas.com/en/tools/

Contents

Section 1 Overview

1.1 Overview

The stub generator (hereafter referred to as RPCGEN) is a tool used to generate the source code of the client and server stubs that are required whenever the remote procedure call (RPC) facility of the HI7200/MP realtime operating system is to be used. A config file is created by using a text editor and then input to RPCGEN, which creates the source code of the client and server stubs.

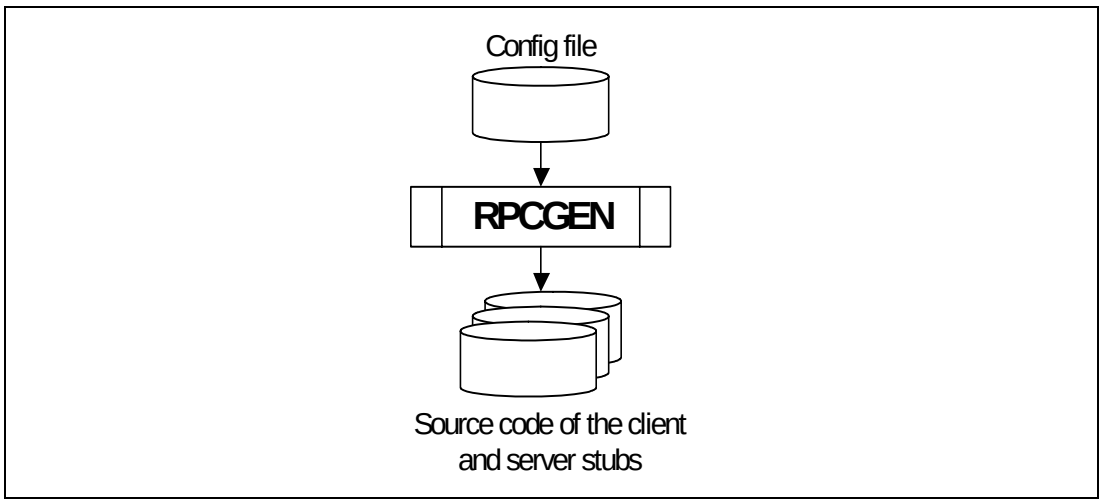

Figure 1.1 Overview of RPCGEN

1.2 Operating Environment

RPCGEN is provided as a Perl script file. Table 1.1 shows the operating environment.

Table 1.1 Operating Environment

Item	Operating Environment
Realtime OS	HI7200/MP V.1.00 Release 00 or later
Perl environment	We have confirmed correct execution of RPCGEN by ActivePerl 5.8.8.820 (for Windows® (x86)) produced by ActiveState Software Inc. ActivePerl is available for free download from the following Web site: http://www.activestate.com/Products/activeperl/
Host machine	Any machine with Windows® XP (32 bits) or Windows® 2000 as the operating system

Section 2 Installation

2.1 Downloading

RPCGEN can be downloaded for free from the Renesas Web site at the following URL.

http://www.renesas.com

2.2 Installing RPCGEN

The file for downloading has been compressed in the zip format. Expanding the zip file generates the files listed in table 2.1. These files must be stored in a suitable folder.

Table 2.1 Files Provided as RPCGEN

Filename	Description
rpcgen.pl	Main body of RPCGEN
rpcgen_clnt.pm	Package modules for use by the main body of RPCGEN
rpcgen_svr.pm	
rpcgen_info.pm	
rpcgen_misc.pm	

Section 3 Functions and Files Generated by RPCGEN

3.1 Functions Generated by RPCGEN

Figure 3.1 shows the functions generated by RPCGEN.

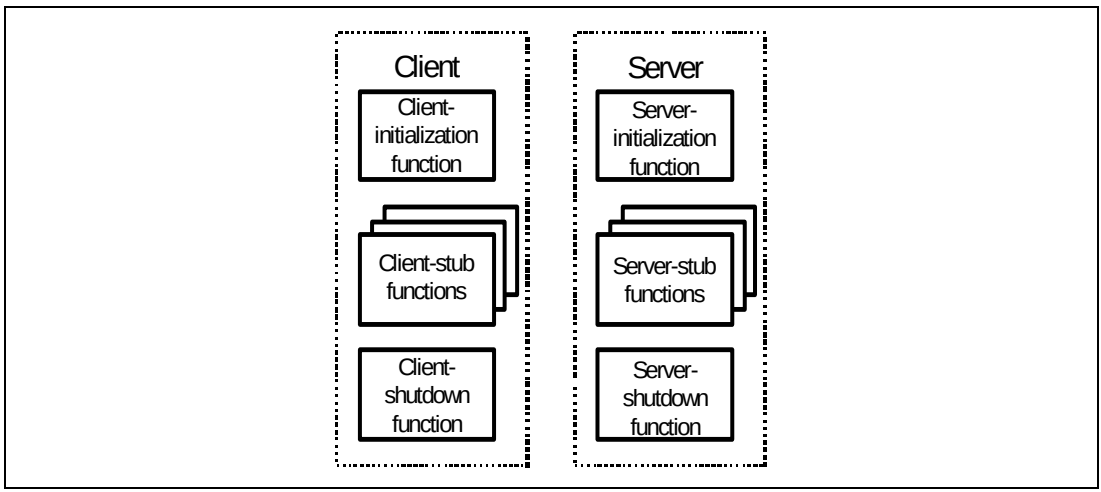

Figure 3.1 Functions Generated by RPCGEN

3.1.1 Client

(1) Client-initialization function

This function initiates the connection with the server. Specifying CLNT_NOINIT prevents the creation of the client-initialization function.

(2) Client-shutdown function

This function ends the connection with the server. Specifying CLNT_NOSHUTDOWN prevents the creation of the client-shutdown function.

(3) Client-stub functions

Each of these functions makes an RPC call and has the same API as or an API similar to the server function. If SVRAUTH has not been specified, the API will be the same as that of the server function. Specifying CLNTSTUB prevents the creation of a specific client-stub function.

3.1.2 Server

(1) Server-initialization function

This function starts up the server. Specifying SVR_NOINIT prevents the creation of the server-initialization function.

(2) Server-shutdown function

This function shuts the server down. Specifying SVR_NOSHUTDOWN prevents the creation of the server-shutdown function.

(3) Server-stub functions

Each of these functions is called back from the RPC library and then calls a server function. Specifying SVRSTUB prevents the creation of a specific server-stub function, while specifying SVRFUNC allows the name of the called server function to be replaced.

3.2 Files Generated by RPCGEN

RPCGEN generates the files listed in table 3.1 in accord with the name of the config file.

Table 3.1 Files Generated by RPCGEN

Filename	Description	Definition of the File Location
<config file>_clnt.c	Client-stub source file	CLNTS option or CLNT_SOURCEPATH in the config file
<config file>_clnt.h	Client-stub header file	CLNTI option or CLNT_INCPATH in the config file
<config file>_private.h	Internal header file for the client stub	
<config file>_svr.c	Server-stub source file	SVRS option or SVR_SOURCEPATH in the config file
<config file>_svr.h	Server-stub header file	SVRI option or SVR_INCPATH in the config file
<config file>_public.h	Header file for the client and server stubs	PUBI option or PUB_INCPATH in the config file

When the config file is “sample.x,” for example, RPCGEN will generate the following files: sample_clnt.c, sample_clnt.h, sample_svr.c, sample_svr.h, sample_private.h, and sample_public.h.

To temporarily save information during the process of file generation, RPCGEN also generates intermediate files in the current directory. The names of these intermediate files consist of two underscores (“__”) appended before the filenames given in the table above.

(1) <config file>_clnt.c (client-stub source file)

This file contains the client-stub functions, client-initialization function, and client-shutdown function.

(2) <config file>_clnt.h (client-stub header file)

This file contains definitions required for use of the client-stub functions, client-initialization function, and client-shutdown function. The definitions include prototype declarations of the client-stub functions, client-initialization function, and client-shutdown function.

(3) <config file>_private.h (internal header file for the client stub)

This is a header file included by <config file>_clnt.c only.

(4) <config file>_svr.c (server-stub source file)

This file contains the server-stub functions, server-initialization function, server-shutdown function, and server-stub function table.

(5) <config file>_svr.h (server-stub header file)

This file contains definitions required for use of the server-initialization function and server-shutdown function. The definitions include prototype declarations of the server-stub functions, server-initialization function, and server-shutdown function.

(6) <config file>_public.h (header file for the client and server stubs)

This is a header file for the client and server applications. For details, refer to section 8.9, <config file>_public.h.

Section 4 Executing RPCGEN

4.1 Executing RPCGEN

RPCGEN must be executed from the command prompt as follows.

```
perlΔ-I<rpcgen path>Δ<rpcgen path>\rpcgen.plΔ
<config file>[Δ<option>...](RET)
```

<rpcgen path> is a path to the directory where RPCGEN has been installed.

<config file> refers to a file describing the specifications of the client and server stubs to be generated. This file must be created by the user. The standard filename extension for the config file is ".x", which must not be omitted when the name is entered in a command line.

After execution of the above command, RPCGEN will generate the C-language source files and header files for the client and server stubs in accord with the information contained in the config file.

If <config file> and <option> are omitted, the command syntax will be shown.

4.2 Options

4.2.1 PUBI

Format:

```
PUBI="<path>"
```

Description:

This option is used to specify the path for output of <config file>_public.h. If this option is used, a definition of PUB_INCPATH in the config file is ignored.

4.2.2 CLNTS

Format:

```
CLNTS="<path>"
```

Description:

This option is used to specify the path for output of <config file>_clnt.c. If this option is used, a definition of CLNT_SOURCEPATH in the config file is ignored.

4.2.3 CLNTI

Format:

```
CLNTI="<path>"
```

Description:

This option is used to specify the path for output of <config file>_clnt.h and <config file>_private.h. If this option is used, a definition of CLNT_INCPATH in the config file is ignored.

4.2.4 SVRS

Format:

```
SVRS="<path>"
```

Description:

This option is used to specify the path for output of <config file>_svr.c. If this option is used, a definition of SVR_SOURCEPATH in the config file is ignored.

4.2.5 SVRI

Format:

```
SVRI="<path>"
```

Description:

This option is used to specify the path for output of <config file>_svr.h. If this option is used, a definition of SVR_INCPATH in the config file is ignored.

4.3 Executing RPCGEN as a Build Phase in the High-performance Embedded Workshop

After RPCGEN has been registered as a custom build phase in the workspace, it will be automatically executed at the time of building.

(1) Defining Custom Placeholders

The following two custom placeholders must be defined.

- Path to RPCGEN
  [Placeholder]: RPCGEN_INST
  [Description]: RPCGEN Base directory
  [Directory]: Directory under which RPCGEN is stored
- Path to Perl.exe
  [Placeholder]: PERL_INST
  [Description]: Perl Base directory
  [Directory]: Directory under which perl.exe is stored

To define a custom placeholder, follow the procedure below.

Select [Setup->Customize] from the menu bar of the High-performance Embedded Workshop. Then select the [Placeholders] tab of the [Customize] dialog box. Choose either [Application wide custom placeholders] or [Workspace wide custom placeholders] and click on the [Add] button. This opens the [New Custom Placeholder] dialog box shown in figure 4.1. Define the two custom placeholders in this dialog box by filling in the information given above.

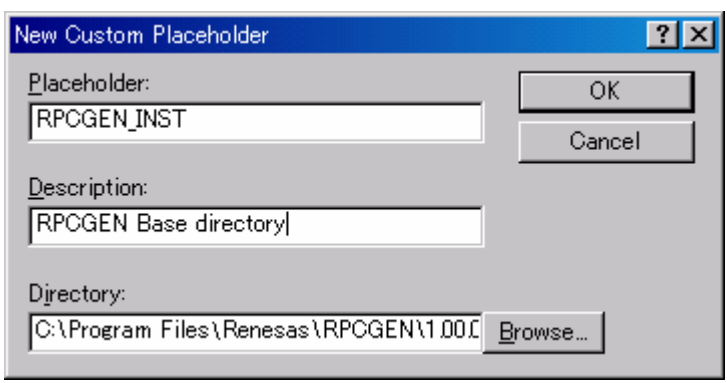

Figure 4.1 [New Custom Placeholder] Dialog Box

(2) Adding a File Extension

To make execution as a custom build phase work, the filename extension ".x" for config files must be added as a filename extension to be used in custom build phases. Select [Project->File Extensions] from the menu bar of the High-performance Embedded Workshop. This opens the [File Extensions] dialog box shown in figure 4.2.

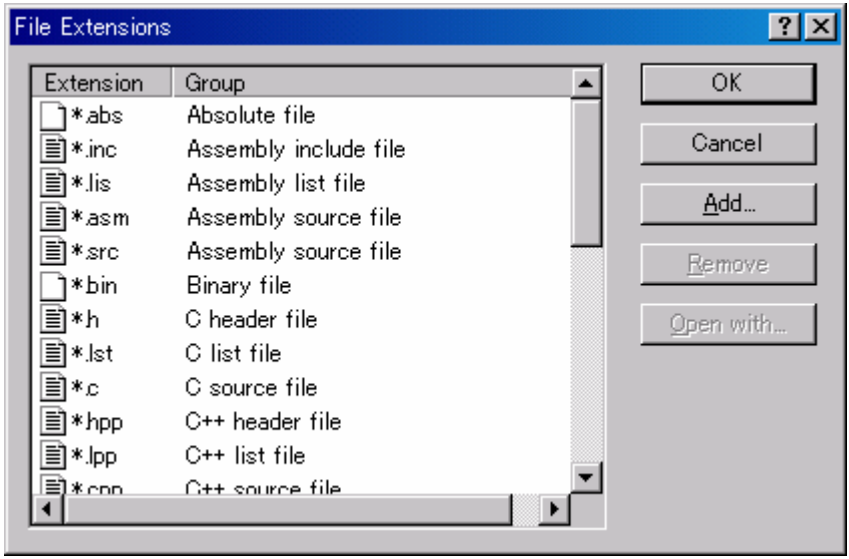

Figure 4.2 [File Extensions] Dialog Box

Clicking the [Add] button opens the [Add File Extension] dialog box shown in figure 4.3. Add the extension ".x" as shown in the figure.

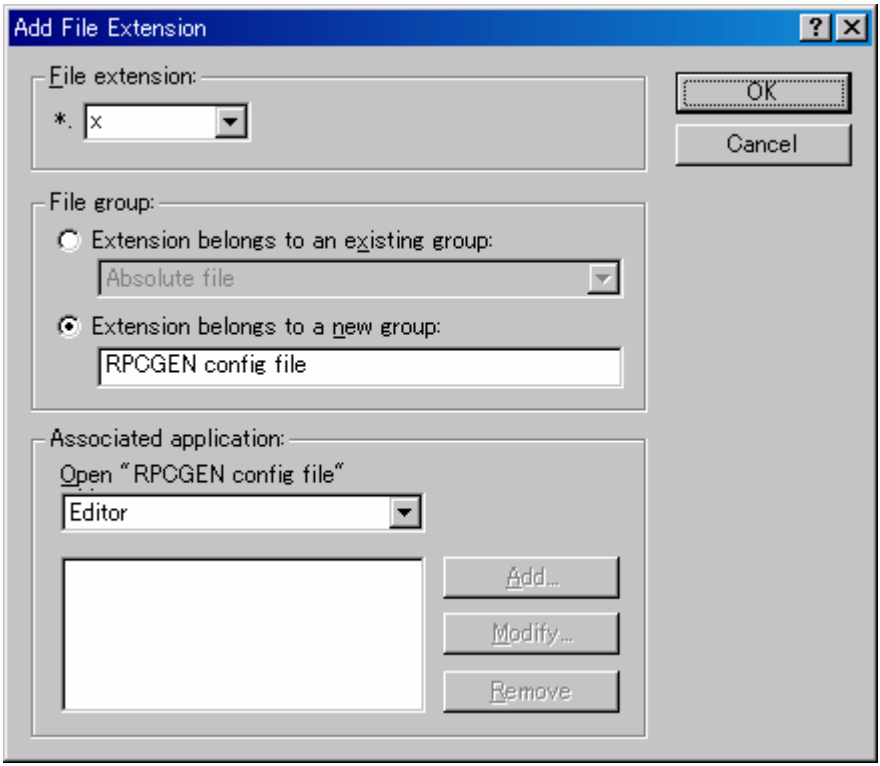

Figure 4.3 [Add File Extension] Dialog Box

(3) Creating the RPCGEN Custom Build Phase

Select [Build->Build Phases] from the menu bar of the High-performance Embedded Workshop. This opens the [Build Phases] dialog box shown in figure 4.4.

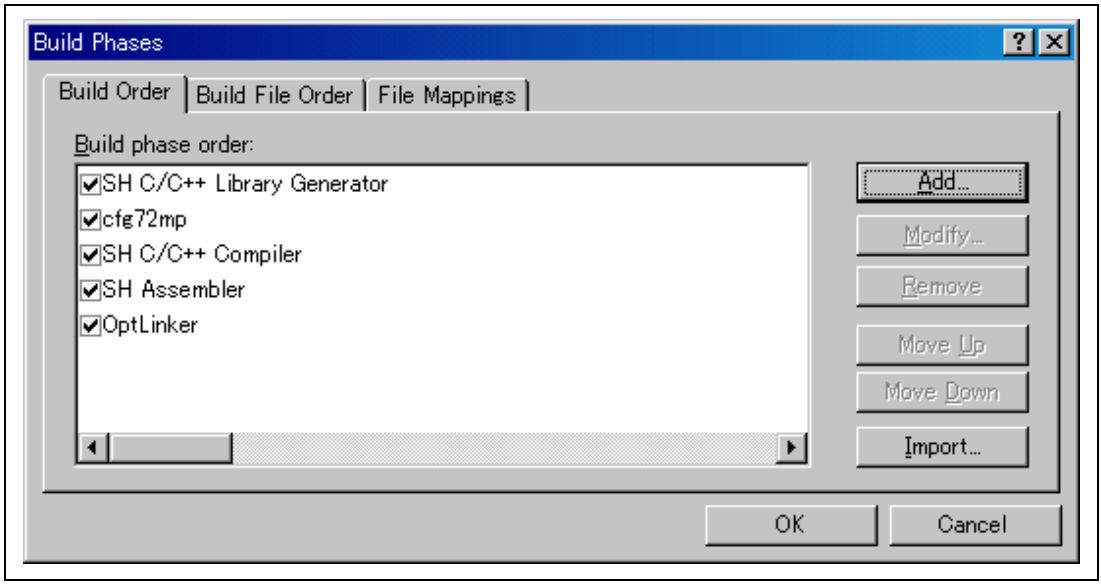

Figure 4.4 [Build Phases] Dialog Box

Clicking on the [Add] button invokes the [New Build Phase] wizard (figure 4.5).

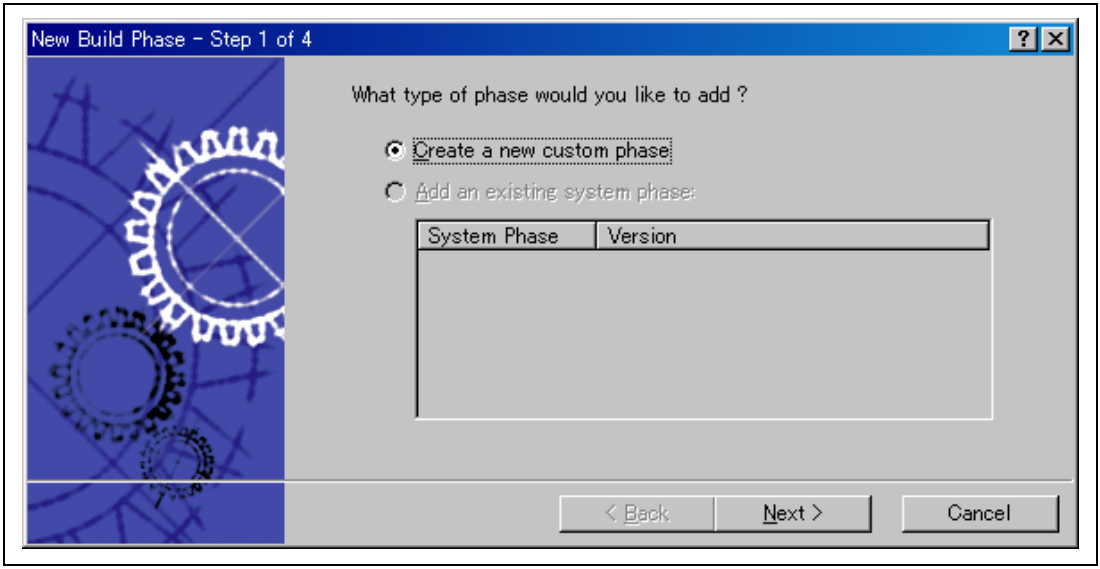

Figure 4.5 [New Build Phase - Step 1 of 4] Dialog Box

Click on [Next] to go to step 2.

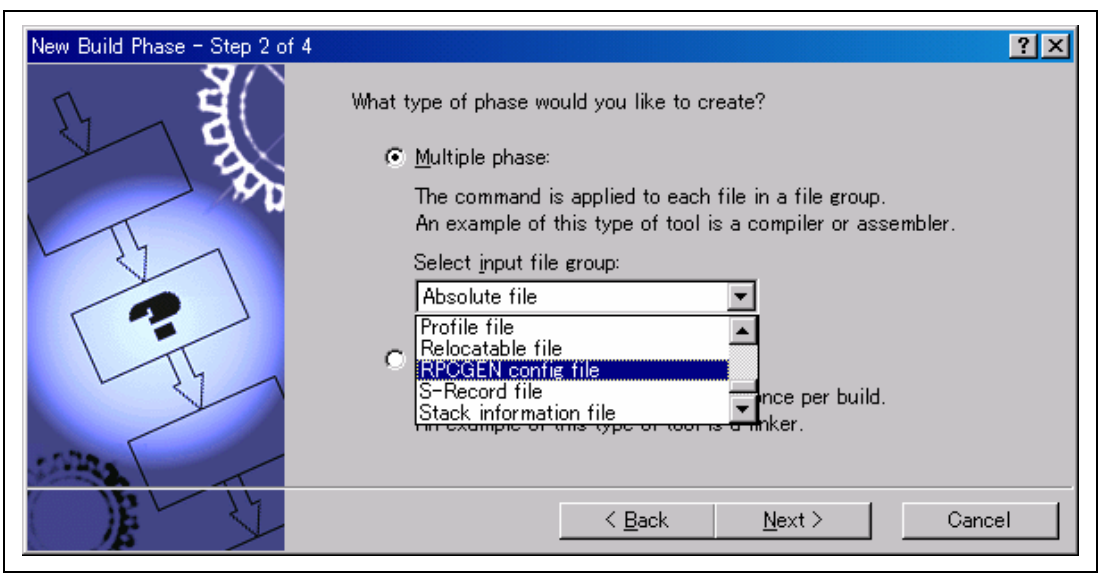

Figure 4.6 [New Build Phase - Step 2 of 4] Dialog Box

At step 2, select [Multiple phase]. Then select [RPCGEN config file] for [Select input file group]. Click on [Next] to go to step 3.

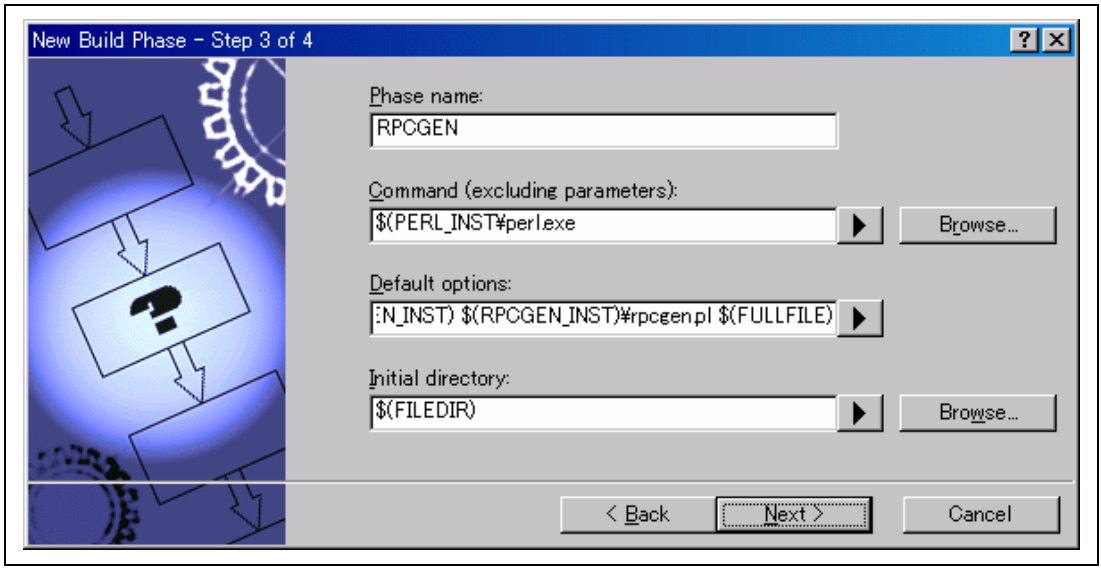

Figure 4.7 [New Build Phase - Step 3 of 4] Dialog Box

[Phase name]: Any name defined by the user (“rpcgen” in this example)
[Command]: $(PERL_INST)\perl.exe
[Default options]: -I $(RPCGEN_INST) $(RPCGEN_INST)\rpcgen.pl $(FULLFILE)
[Initial directory]: $(FILEDIR)

Click on [Next] to go to step 4.

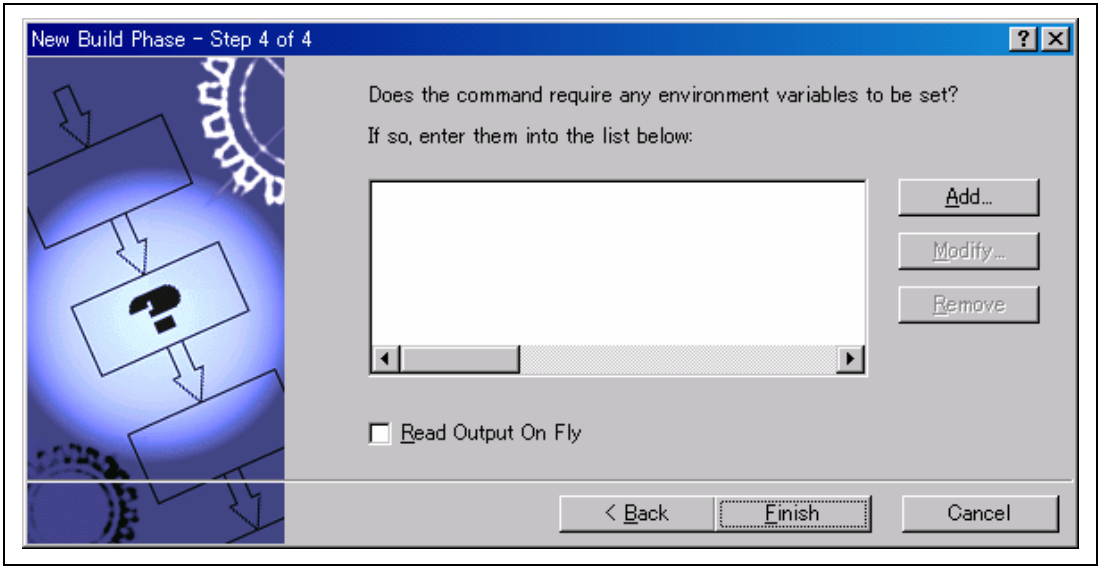

Figure 4.8 [New Build Phase - Step 4 of 4] Dialog Box

Although no environment variables need to be added for RPCGEN, you can add environment variables for the Perl environment as required.

Creation of the RPCGEN custom build phase is now complete.

Next, the syntax for messages about RPCGEN must be defined. After the syntax has been defined, the corresponding config file (*.x) will be opened in response to double-clicking on an RPCGEN error message output in the [Build] window of the High-performance Embedded Workshop.

In the [Build Phases] dialog box shown in figure 4.4, select “rpcgen” and click on the [Modify] button. The [Modify rpcgen] dialog box opens. Select the [Output Syntax] tab and type in the syntax for errors shown in figure 4.9.

Figure 4.9 [Add Output Syntax] Dialog Box

Now settings must be made so that the files output by RPCGEN will be deleted by [Build->Clean Current Project] or [Build->Clean All Projects] from the High-performance Embedded Workshop.

Select [Build->rpcgen] to open the [rpcgen Options] dialog box (figure 4.10). Select the [Output Files] tab, check that the “RPCGEN config file” folder icon is active, and click on the [Add] button. This opens the [Add Output File] dialog box, in which the names of the files output by RPCGEN should be entered. For the filenames, refer to section 3.2, Files Generated by RPCGEN.

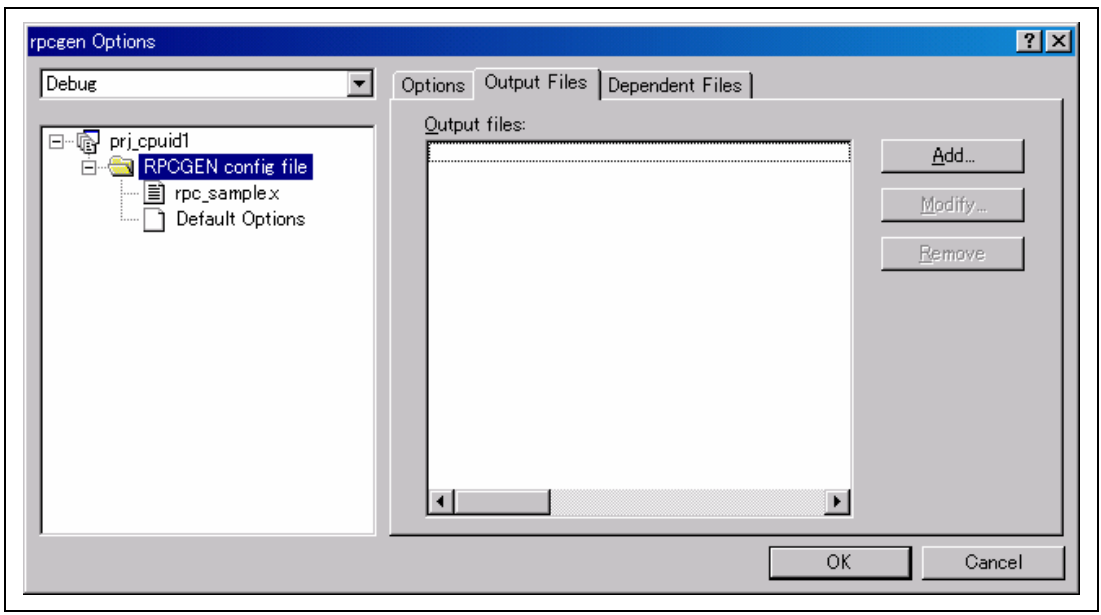

Figure 4.10 [rpcgen Options] Dialog Box

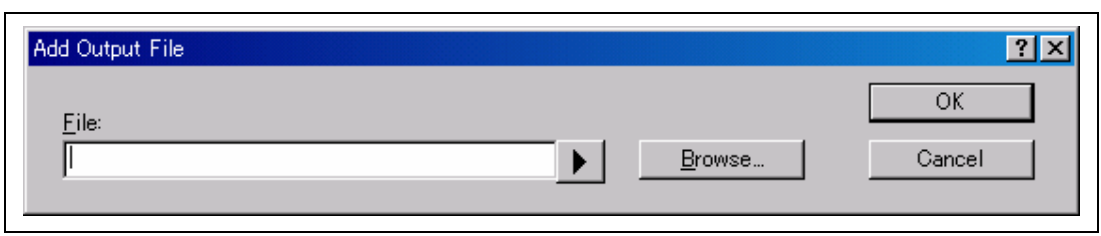

Figure 4.11 [Add Output File] Dialog Box

(4) Settings in the [Build Phases] Dialog Box

The created RPCGEN custom build phase is shown at the bottom of the [Build Phases] dialog box. Use the [Move Up] button to move “rpcgen” so that it is above “SH C/C++ Library Generator.”

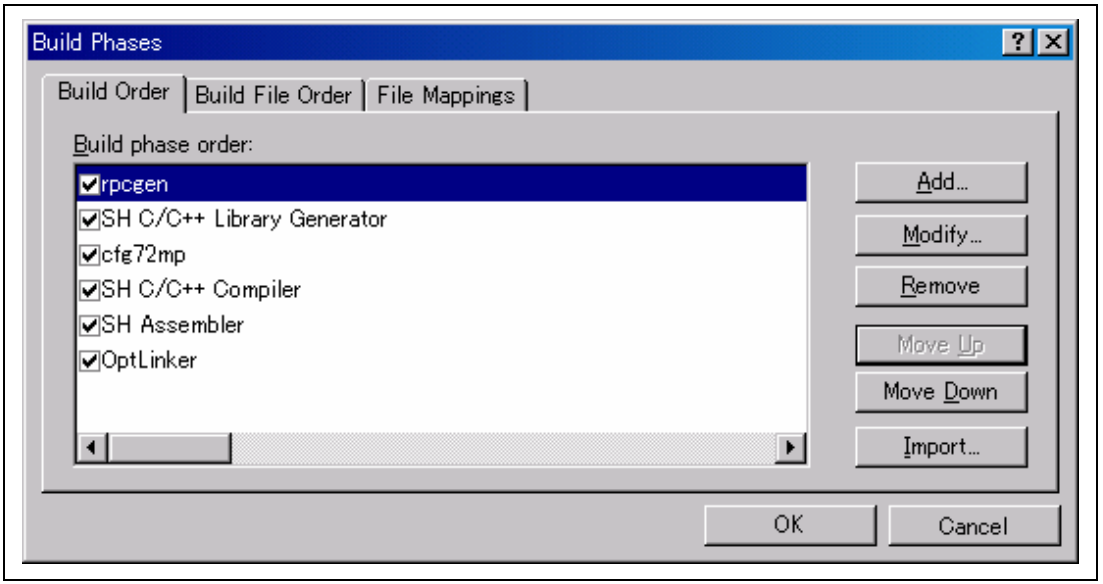

Figure 4.12 [Build Phases] Dialog Box - [Build Order] Tab

Open the [Build File Order] tab. Select “RPCGEN config file” under [File group] and check the box for [rpcgen] under [Phase order].

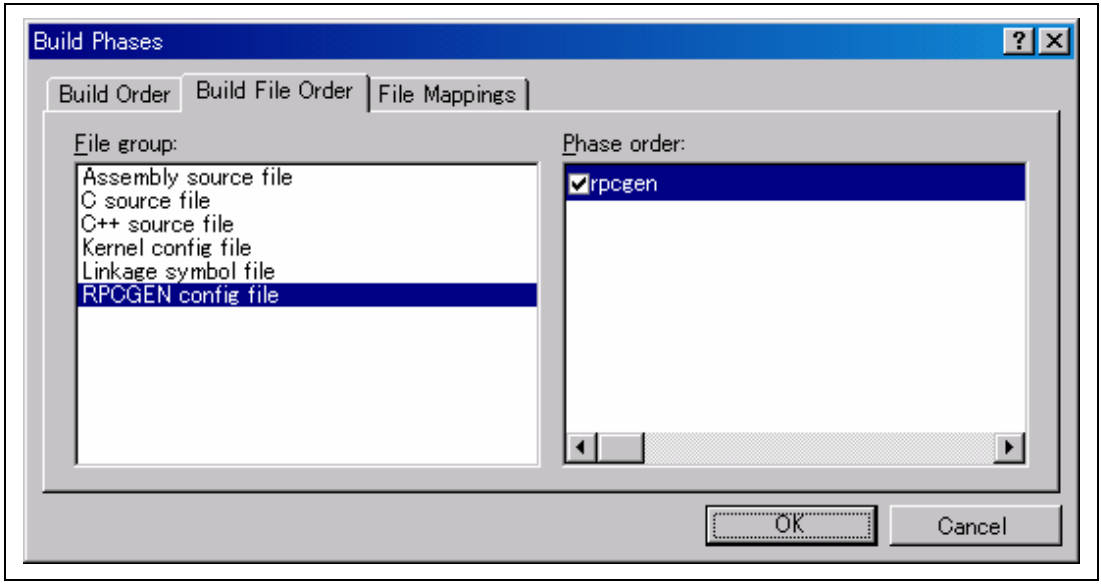

Figure 4.13 [Build Phases] Dialog Box - [Build File Order] Tab

Setting of the RPCGEN build phase is now complete.

Section 5 Specifications of a Config File

A config file contains definitions used for creation of the client and server stubs. Specifically, such definitions are made by appending keywords for passing to RPCGEN to the API of the server functions (as described in section 6, Definitions of Server Functions). Read this section carefully before creating your own config file. The extension for config files is ".x."

One config file corresponds to the definition of one server. Do not attempt to define multiple servers within a single config file.

5.1 Definition

Definitions must be written in either of the following formats.

```
<keyword>{...};
<keyword>;
```

For certain definitions, two or more items can be entered between the braces {...}.

Example:

```
GLOBAL_INCFILE { "types.h" "rpc_public.h" };
```

A config file can include the following definitions.

(1) Locations where the files created by RPCGEN will be stored

- PUB_INCPATH{…}; Path to <config file>_public.h
- CLNT_SOURCEPATH{…}; Path to <config file>_clnt.c
- CLNT_INCPATH{…}; Path to <config file>_clnt.h and <config file>_private.h
- SVR_SOURCEPATH{…}; Path to <config file>_svr.c
- SVR_INCPATH{…}; Path to <config file>_svr.h

(2) Controlling output of the #include directive to the files created by RPCGEN

- GLOBAL_INCFILE{…}; Include file common to the client and server stubs
- CLNT_INCFILE{…}; Include file for the client stub
- SVR_INCFILE{…}; Include file for the server stub

(3) Server information

- SVR_NAME{…}; Server name
- SVR_ID{…}; Server ID
- SVR_VERSION{…}; Server version
- SVR_NOINIT; The server-initialization function is not to be created.
- SVR_NOSHUTDOWN; The server-shutdown function is not to be created.
- SVR_NOSTUBTBL; The server-stub function table is not to be created.
- SVR_STATIC{…}; Use of a static server
- SVR_SECTION{…}; Section name to be given to the server stub
- SVR_AUTH; How the server ID and server version will be assigned

(4) Client information

- CLNT_NOINIT; The client-initialization function is not to be created.
- CLNT_NOSHUTDOWN; The client-shutdown function is not to be created.
- CLNT_CALLCHK; Facilitates saving the return values of RPC calls
- CLNT_SECTION{…}; Section name to be given to the client stub

(5) Server functions

- RPC_FUNC{…}; Server functions

Server functions are only defined within the braces of RPC_FUNC; multiple server functions can be defined.

5.2 Comments

Comments in config files should be written in the same way as in the C and C++ languages. The comments that are written before the first valid definition of a keyword will be output to the beginning of all source and header files generated by RPCGEN.

5.3 File Locations

Use the statements listed below to define paths to the files generated by RPCGEN. If these definitions are omitted, RPCGEN will generate the source or header files in the current directory.

If a definition is otherwise correct but the defined path does not exist or the directory already contains read-only files, RPCGEN displays an error message and terminates the processing.

The following definitions of file locations are available.

- PUB_INCPATH{…}; Path to <config file>_public.h
- CLNT_SOURCEPATH{…}; Path to <config file>_clnt.c
- CLNT_INCPATH{…}; Path to <config file>_clnt.h and <config file>_private.h
- SVR_SOURCEPATH{…}; Path to <config file>_svr.c
- SVR_INCPATH{…}; Path to <config file>_svr.h

5.3.1 PUB_INCPATH

Format:

```
PUB_INCPATH{"<path>"};
```

Description:

This statement defines the path under which <config file>_public.h will be generated. When the PUBI option is used, a definition of PUB_INCPATH is ignored.

Example:

```
PUB_INCPATH{"public_include"};
```

<config file>_public.h will be generated in the public_include directory under the current directory.

5.3.2 CLNT_SOURCEPATH

Format:

```
CLNT_SOURCEPATH{"<path>"};
```

Description:

This statement defines the path under which <config file>_clnt.c will be generated. When the CLNTS option is used, a definition of CLNT_SOURCEPATH is ignored.

Example:

```
CLNT_SOURCEPATH{"clnt\source"};
```

<config file>_clnt.c will be generated in the clnt\source directory under the current directory.

5.3.3 CLNT_INCPATH

Format:

```
CLNT_INCPATH{"<path>"};
```

Description:

This statement defines the path under which <config file>_clnt.h and <config file>_private.h will be generated. When the CLNTI option is used, a definition of CLNT_INCPATH is ignored.

Example:

```
CLNT_INCPATH{"C:\clnt\include"};
```

<config file>_clnt.h and <config file>_private.h will be generated in the C:\clnt\include directory.

5.3.4 SVR_SOURCEPATH

Format:

```
SVR_SOURCEPATH{"<path>"};
```

Description:

This statement defines the path under which <config file>_svr.c will be generated. When the SVRS option is used, a definition of SVR_SOURCEPATH is ignored.

Example:

```
SVR_SOURCEPATH{"svr\source"};
```

<config file>_svr.c will be generated in the svr\source directory under the current directory.

5.3.5 SVR_INCPATH

Format:

```
SVR_INCPATH{"<path>"};
```

Description:

This statement defines the path under which <config file>_svr.h will be generated. When the SVRI option is used, a definition of SVR_INCPATH is ignored.

Example:

```
SVR_INCPATH{"svr\include"};
```

<config file>_svr.h will be generated in the svr\include directory under the current directory.

5.4 Controlling the Output of #include Directives

Use the statements listed below to output #include directives for user files to the files generated by RPCGEN.

Note, however, that #include directives for the following include files will always be output.

Table 5.1 Default Include Files

Filename	Description
string.h	Standard include file for the compiler
types.h	Header file of data-type definitions; comes with Renesas operating systems (such as the HI7200/MP)
rpc_public.h	RPC header file that comes with Renesas operating systems (such as the HI7200/MP)

The following definitions are available for controlling the output of #include directives.

- GLOBAL_INCFILE{…}; Include file common to the client and server stubs
- CLNT_INCFILE{…}; Include file for the client stub
- SVR_INCFILE{…}; Include file for the server stub

5.4.1 Order of #Include Directives

RPCGEN outputs #include directives to files in the following order.

(1) <config file>_clnt.c

1. Default include files (in the order given in table 5.1)
2. File(s) defined with GLOBAL_INCFILE (in the order of file definition)
3. File(s) defined with CLNT_INCFILE (in the order of file definition)
4. <config file>_public.h generated by RPCGEN (only if SVR_AUTH has not been defined)
5. <config file>_clnt.h generated by RPCGEN
6. <config file>_private.h generated by RPCGEN

(2) <config file>_svr.c

1. Default include files (in the order given in table 5.1)
2. File(s) defined with GLOBAL_INCFILE (in the order of file definition)
3. File(s) defined with SVR_INCFILE (in the order of file definition)
4. <config file>_public.h generated by RPCGEN (only if SVR_AUTH has not been defined)
5. <config file>_svr.h generated by RPCGEN

5.4.2 GLOBAL_INCFILE

Format:

```
GLOBAL_INCFILE{<file>[Δ<file>...]};
```

Description:

This statement is for the output of #include directives for the specified files to <config file>_clnt.c and <config file>_svr.c. File(s) must be specified in the form "filename" or <filename>. The difference between these two formats is as defined in the specification of the #include directive.

When two or more files are to be specified, they must be separated by a space.

Example:

```
GLOBAL_INCFILE{<math.h> "import\include\user_public.h"};
```

5.4.3 CLNT_INCFILE

Format:

```
CLNT_INCFILE{<file>[Δ<file>...]};
```

Description:

This statement is for the output of #include directives for the specified files to <config file>_clnt.c. File(s) must be specified in the form "filename" or <filename>. The difference between these two formats is as defined in the specification of the #include directive.

When two or more files are to be specified, they must be separated by a space.

Example:

```
CLNT_INCFILE{<math.h> "import\include\user_clnt.h"};
```

5.4.4 SVR_INCFILE

Format:

```
SVR_INCFILE{<file>[Δ<file>...]};
```

Description:

This statement is for the output of #include directives for the specified files to <config file>_svr.c. File(s) must be specified in the form "filename" or <filename>. The difference between these two formats is as defined in the specification of the #include directive.

When two or more files are to be specified, they must be separated by a space.

Example:

```
SVR_INCFILE{<math.h> "import\include\user_svr.h"};
```

5.5 Server Information

The following types of definitions are available for defining server information.

- SVR_NAME{…}; Server name
- SVR_ID{…}; Server ID
- SVR_VERSION{…}; Server version
- SVR_NOINIT; The server-initialization function is not to be created.
- SVR_NOSHUTDOWN; The server-shutdown function is not to be created.
- SVR_NOSTUBTBL; The server-stub function table is not to be created.
- SVR_STATIC{…}; Use of a static server
- SVR_SECTION{…}; Section name to be given to the server stub
- SVR_AUTH; How the server ID and server version will be assigned

5.5.1 SVR_NAME

Format:

```
SVR_NAME {<server name>};
```

Description:

This statement defines the server name. The defined server name will be applied to the names of stub functions, etc. generated by RPCGEN. This definition cannot be omitted; if it is omitted, RPCGEN reports the error and terminates processing.

Example:

```
SVR_NAME{EXAMPLE};
```

5.5.2 SVR_ID

Format:

```
SVR_ID {<ID number>};
```

Description:

This statement defines the server ID. The ID number can only be specified as a 4-byte integer constant that can be represented as UINT32 or a C-language macro name to be replaced by such a 4-byte integer constant. If a C-language macro is used, this macro must have been defined in a file specified by GLOBAL_INCFILE.

The server ID must, of course, be consistent across the entire system due to the specifications of the HI7200/MP. RPCGEN does not detect duplication of server IDs.

The ID number is output to <config file>_public.h in the following format.

```
#define  RPCSVR_ID_<server name>  <ID number>
```

<Server name> refers to the server name defined with SVR_NAME.

SVR_ID cannot be omitted if a definition of SVR_AUTH has been omitted. Also refer to section 5.5.8, SVR_AUTH.

Example:

```
SVR_ID{1UL};
```

5.5.3 SVR_VERSION

Format:

```
SVR_VERSION {<server version>};
```

Description:

This statement defines the server version. The server version can only be specified as a 4-byte integer constant that can be represented as UINT32 or a C-language macro name to be replaced by such a 4-byte integer constant. If a C-language macro is used, this macro must have been defined in a file specified by GLOBAL_INCFILE.

The server version is output to <config file>_public.h in the following format.

```
#define  RPCSVR_VERS_<server name>  <server version>
```

SVR_VERSION cannot be omitted if SVR_AUTH has been omitted. Also refer to section 5.5.8, SVR_AUTH.

Example:

```
SVR_VERSION{1UL};
```

5.5.4 SVR_NOINIT

Format:

```
SVR_NOINIT;
```

Description:

If this statement is present, RPCGEN will not create the server-initialization function. Use SVR_NOINIT when the server-initialization function is provided as part of the application. Also refer to sections 8.3, Server-Initialization Function, and 8.4, Server-Stub Function Table.

Example:

```
SVR_NOINIT;
```

5.5.5 SVR_NOSTUBTBL

Format:

```
SVR_NOSTUBTBL;
```

Description:

The server-stub function table is rpc_server_info.ServerStubList, the table defined by rpc_start_server() or rpc_start_server_with_paramarea() called by the server-initialization function.

If this statement is present, RPCGEN will not create the server-stub function table. Use SVR_NOSTUBTBL when the server-initialization function is provided as part of the application. Also refer to section 8.4, Server-Stub Function Table.

Example:

```
SVR_NOSTUBTBL;
```

5.5.6 SVR_NOSHUTDOWN

Format:

```
SVR_NOSHUTDOWN;
```

Description:

If this statement is present, RPCGEN will not create the server-shutdown function. Use SVR_NOSHUTDOWN when the server-shutdown function is provided as part of the application.

The server-shutdown function generated by RPCGEN calls rpc_stop_server() to stop the server. Although rpc_stop_server() generally allows specification of a callback function to be executed at the time the server is stopped, no callback function is specifiable for the server-shutdown function generated by RPCGEN. If you wish to use a callback function, specify SVR_NOSHUTDOWN and implement the server-shutdown function on the user side.

Also refer to section 8.5, Server-Shutdown Function.

Example:

```
SVR_NOSHUTDOWN;
```

5.5.7 SVR_STATIC

Format:

```
SVR_STATIC{<size>Δ<section name>};
```

Description:

This statement defines the server as static. If this definition is omitted, the server will be dynamic. Note that SVR_STATIC cannot be used in conjunction with SVR_NOINIT.

<Size> indicates the size of the server parameter area. Specify an integer constant that can be represented as UINT32. The specified value is rounded up to the nearest multiple of four.

<Section name> indicates the section name to be given to the server parameter area. Note, however, that the actual section name will be 'B' followed by the section name given here. This <section name> is not affected by definition of SVR_SECTION.

The server-parameter area information is output to <config file>_svr.c in the following format.

```
#pragma section <section name>
#pragma pack 4
static UINT8 ucServerArea_<server name>[ ALIGNUP4(<size>) ];
#pragma unpack
#pragma section
```

Note:

Since the SH2A-DUAL does not have a facility for the CPU cores to snoop on each other's caches, the server parameter area must be allocated in a non-cacheable area. Specify a section name that can be distinguished from those of sections to be allocated in cacheable areas.

Example:

```
SVR_STATIC{ 512  D_SVRAREA};
```

5.5.8 SVR_AUTH

Format:

```
SVR_AUTH;
```

Description:

There are two ways to assign the server ID and server version.

(1) Model 1

Model 1 applies when SVR_AUTH has not been specified.

In this model, the application does not recognize the server ID or server version. The server ID and server version are determined within the client and server stubs generated by RPCGEN.

Each of the client-stub functions has the same API as the original server function. Server IDs and server version information passed to the following functions will be ignored.

- Server-initialization function
- Server-shutdown function
- Client-initialization function
- Client-shutdown function

RPCGEN outputs the definitions of the server ID and server version to <config file>_public.h.

Since the client-stub functions and server- and client- initialization and shutdown functions generated by RPCGEN have include statements for <config file>_public.h, the server ID and server version information is passed to the RPC library.

If RPCGEN-generated files for the client and server are from different generations, the RPC library treats any RPC call as an error.

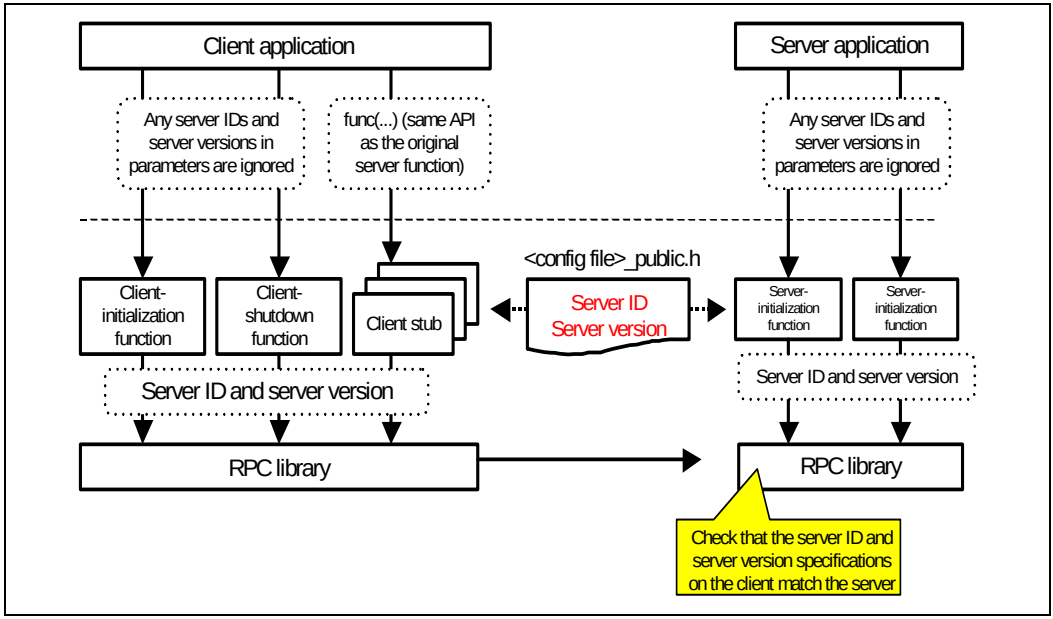

Figure 5.1 Model 1

(2) Model 2

Model 2 applies when SVR_AUTH has been specified.

In this model, the application assigns the server ID and server version. RPCGEN outputs definitions of the server ID and server version in <config file>_public.h so that the application is able to acquire this information.

Each of the client-stub functions is in the form of rpcclnt_<server name>_<server function name> and its first and second parameters are the server ID and server version, respectively. The third and subsequent parameters are the parameters of the actual server function. In other words, all of the client-stub functions have APIs that differ from the server functions. To make the client-stub functions have the same APIs as the original server functions, the user must implement wrapper functions for the client-stub functions.

RPCGEN outputs definitions of the server ID and server version to <config file>_public.h. Applications that call client-stub functions and the server-initialization function must thus include <config file>_public.h so that the server ID and version are specified.

If <config file>_public.h for the client and server are from different generations, the RPC library treats any RPC call as an error.

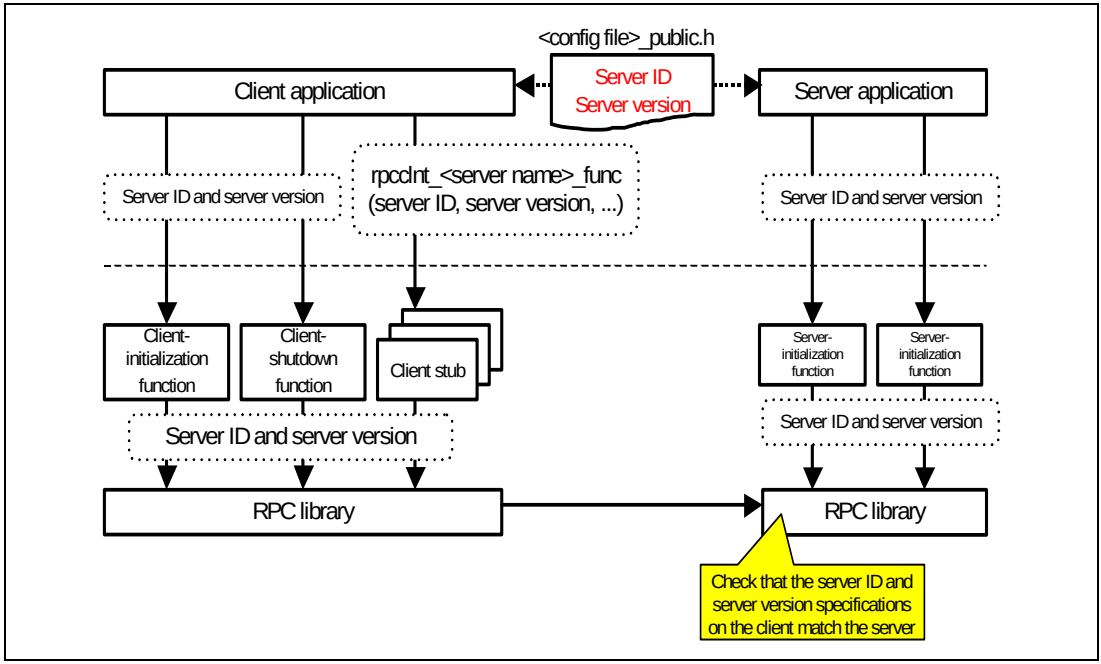

Figure 5.2 Model 2

Example:

```
SVR_AUTH;
```

5.5.9 SVR_SECTION

Format:

```
SVR_SECTION{<section name>};
```

Description:

This statement specifies the section to which the server stubs will be assigned. When this definition is omitted, one of the section names (alphabetical characters) listed below will automatically be assigned. If SVR_SECTION has been defined, the actual section name will be one of the letters listed below followed by <section name>.

- Program section: ‘P’
- Constant section: ‘C’
- Non-initialized data section: ‘B’
- Initialized data section: ‘D’

Example:

```
SVR_SECTION{ C_EXAMPLE };
```

5.6 Client Information

The following definitions of client information are available.

- CLNT_NOINIT; The client-initialization function is not to be created.
- CLNT_NOSHUTDOWN; The client-shutdown function is not to be created.
- CLNT_CALLCHK; Facilitates saving the return value in RPC calls
- CLNT_SECTION{…}; Section name to be given to the client stub

5.6.1 CLNT_NOINIT

Format:

```
CLNT_NOINIT;
```

Description:

RPCGEN creates the following client-initialization function in <config file>_clnt.c as the standard:

```
INT32 rpcclnt_<server name>_init(UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion );
```

If the CLNT_NOINIT statement is present, RPCGEN will not create the client-initialization function. Use CLNT_NOINIT when the client-initialization function is provided as part of the application. Also refer to section 8.6, Client-Initialization Function.

Example:

```
CLNT_NOINIT;
```

5.6.2 CLNT_NOSHUTDOWN

Format:

```
CLNT_NOSHUTDOWN;
```

Description:

RPCGEN creates the following client-shutdown function in <config file>_clnt.c as the standard:

```
INT32 rpcclnt_<server name>_shutdown(UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion );
```

If the CLNT_NOSHUTDOWN statement is present, RPCGEN will not create the client-shutdown function. Use CLNT_NOSHUTDOWN when the client-shutdown function is provided as part of the application.

The client-shutdown function generated by RPCGEN calls rpc_disconnect() to disconnect the client from the server. Although rpc_disconnect() generally allows specification of a callback function to be executed at the time the client is disconnected from the server, no callback function is specifiable for the client-shutdown function generated by RPCGEN. If you wish to use a callback function, specify CLNT_NOSHUTDOWN and implement the client-shutdown function on the user side.

In the current HI7200/MP specification, a callback function is ignored even if one has been specified for rpc_disconnect().

Also refer to section 8.7, Client-Shutdown Function.

Example:

```
CLNT_NOSHUTDOWN;
```

5.6.3 CLNT_CALLCHK

Format:

```
CLNT_CALLCHK;
```

Description:

RPCGEN creates the following code in <config file>_clnt.c to facilitate saving of the return values of RPC calls (rpc_call() or rpc_call_copycbk()) and outputs the API of *rpc_retval_adr() in <config file>_clnt.h.

```
*rpc_retval_adr() = rpc_call(...);
```

rpc_retval_adr() must be implemented by the user. If this definition is omitted, the following code, which discards the return value of an RPC call, will be created.

```
rpc_call(...);
```

For how to implement rpc_retval_adr(), refer to section 8.8, rpc_retval_adr().

Example:

```
CLNT_CALLCHK;
```

5.6.4 CLNT_SECTION

Format:

```
CLNT_SECTION{<section name>};
```

Description:

This statement specifies the section to which the client stubs will be assigned. When this definition is omitted, one of the section names (alphabetical characters) listed below will automatically be assigned. If CLNT_SECTION has been defined, the actual section name will be one of the letters listed below followed by the <section name>.

- Program section: ‘P’
- Constant section: ‘C’
- Non-initialized data section: ‘B’
- Initialized data section: ‘D’

Example:

```
CLNT_SECTION{ C_EXAMPLE };
```

5.7 Server Functions

The following definition statement is available for defining server functions.

- RPC_FUNC{…}; Server functions

5.7.1 RPC_FUNC

Format:

```
RPC_FUNC
{
  <definitions of server functions>;
  ...
};
```

Description:

Enter definitions of server functions between the braces {} of RPC_FUNC. For information on definition statements for server functions, refer to section 6, Definitions of Server Functions.

Example:

```
RPC_FUNC
{
  int ret = func1([IN DFLT]int par);
  func2([OUT PTR]struct ST *ptr);
};
```

Section 6 Definitions of Server Functions

6.1 Format

Server functions must be defined in one of the following formats.

```
(1) <specification of the return value> <variable to be returned> =
<function name>(<parameter>, ...)[<option>Δ...];
(2) <function name> (<parameter>, ...)[<option>Δ...];
(3) <directive to extend the return value><specification of the return
value> <variable to be returned> = <function name>(<parameter>, ...)
[<option>Δ...];
```

RPCGEN keywords enclosed by [] are specified in <parameter>, <option>, and <directive to extend the return value>.

Example 1:

Specification of a server function

```
int func(int par);
```

The function takes par as an input and returns a value of type int.

Definition of the server function

```
int ret = func([IN DFLT]int par);
```

Description

- Directive to extend the return value: None
- Specification of the return value: int ret =
- Function name: func
- Parameter directive: [IN DFLT]
- Parameter specification: int par
- Option: None

Example 2:

Specification of a server function

```
void func(struct ST *ptr);
```

The function takes ptr, a pointer to a structure of type ST, as an input and does not return a value.

Definition of the server function

```
func([IN PTR]struct ST *ptr)[UNACK];
```

Description

- Directive to extend the return value: None
- Specification of the return value: None
- Function name: func
- Parameter directive: [IN PTR]
- Parameter specification: struct ST *ptr
- Option: [UNACK]

Example 3:

Specification of a server function

```
double func(double inf);
```

The input is inf and a value of type double will be returned.

Definition of the server function

```
[RETEXT]double dret = func([IN DFLT]double inf);
```

Description

- Directive to extend the return value: [RETEXT]
- Specification of the return value: double dret =
- Function name: func
- Parameter directive: [IN DFLT]
- Parameter specification: double inf
- Option: None

For details on each of the descriptions, refer to the following sections.

- Function-type directives: 6.2, Function-Type Directives
- Specifying the return value: 6.2, Function-Type Directives
- Function name: 6.3, Function Names
- Parameter: 6.4, Parameters
- Option: 6.7, Optional Keywords

6.2 Function-Type Directives

6.2.1 Function with a Return Value

Format:

```
<type of the return value>Δ<variable to hold the return value>=
```

Description:

<Variable to hold the return value> refers to a local variable used to store the return value in both client and server stubs. A parameter of the function may be specified as this variable.

Example:

Specification of a server function

```
int func(int par);
```

The function takes par as an input and returns a value of type int.

Definition of the server function

```
int ret = func([IN DFLT]int par);
```

6.2.2 Function without a Return Value

Description:

For a function that doesn't have a return value, a return-value specification must not be made.

Example:

Specification of a server function

```
void func(struct ST *ptr);
```

The function has no input and returns a value in the ST structure indicated by *ptr.

Definition of the server function

```
func([OUT PTR]struct ST *ptr);
```

6.2.3 When the Return Value is not Representable as a 4-Byte Integer

Format:

```
[RETEXT]Δ<type of the return value>Δ
<variable to hold the return value>=
```

Description:

By default, the return value is cast into UINT32 in the server stub and then transferred to the client. This value is cast back into the original type in the client stub and is then returned.

When the return value is of the following types, however, the definition must have an RETEXT directive because the default behavior above is not applicable in these cases.

- 64-bit integer
- Floating point
- Structure-type object
- Union-type object

When a definition has an RETEXT directive, code will be generated such that a value is returned from the server to the client by using the structure IOVEC.

Example:

Specification of a server function

```
double func(int par);
```

The function takes par as an input and returns a value of type double.

Definition of the server function

```
[RETEXT]double ret = func([IN DFLT]int par);
```

6.3 Function Names

Description:

Type the name of the function you wish to use.

Example:

Specification of a server function

```
int func(int par);
```

The function takes par as an input and returns a value of type int.

Definition of the server function

```
int ret = func([IN DFLT]int par);
```

6.4 Parameters

Parameters must be defined in the following format.

```
[parameter directive]<parameter specification>
```

The parameter directive is a declaration of the following parameter specification and has the form given below.

```
[<type of input/output>Δ<data type>]
```

<type of input/output>: Specify one of the keywords given in section 6.5, Keywords for Input/Output.

<data type>: Specify one of the keywords given in section 6.6, Keywords for Data Types.

<parameter specification>: Define the specification of the original function.

Table 6.1 lists the available combinations of keywords although details on input/output and data types will be described in later sections. Note, however, that REF and DESC cannot be used at the same time.

Table 6.1 Input/Output and Data Types

Input/Output		Data Type			
	Option	**DFLT**	**STR**	**PTR**	**ARY**
IN	None	✓	✓	✓	✓
	REF	—	✓	✓	✓
	DESC	✓	✓	✓	✓
OUT	None	—	—	✓	✓
	REF	—	—	—	—
	DESC	—	—	—	—
INOUT	None	—	—	✓	✓
	REF	—	—	—	—
	DESC	—	—	—	—

✓: Available

—: Not available (RPCGEN reports an error and terminates the processing)

6.5 Keywords for Input/Output

The following three keywords are available for setting the input/output attributes of parameters.

- IN: Input
- OUT: Output
- INOUT: Input and output

The following optional keywords can also be attached to the keywords listed above (note, however, that REF and DESC cannot be used at the same time).

- REF: Passing by reference
- DESC: The parameter is not passed from the client to the server

Each of these keywords, specified with one of the data-type keywords listed in section 6.6, works as a declaration of the subsequent parameter specification.

6.5.1 IN (Input)

Description:

This keyword defines a parameter as an input to the server function. The parameter is transferred from the client to the server and is then passed to the server function.

Example:

Specification of a server function

```
int func(int par);
```

The function takes par as an input and returns a value of type int.

Definition of the server function

```
int ret = func([IN DFLT]int par);
```

6.5.2 OUT (Output)

Description:

This keyword defines a parameter as an output from the server function. The data output by the server function to the area specified by the parameter are returned to the client. This keyword can only be used with the data-type keywords PTR and ARY.

Example:

Specification of a server function

```
void func(struct ST *ptr);
```

The function has no input and returns a value in the ST structure indicated by *ptr.

Definition of the server function

```
func([OUT PTR]struct ST *ptr);
```

6.5.3 INOUT (Input and Output)

Description:

This keyword defines a parameter as an input to and output from the server function. The data in the area specified by the parameter are transferred from the client to the server and then passed to the server function. After that, the data output by the server function to the area specified by the parameter are returned to the client. This keyword can only be used with the data-type keywords PTR and ARY.

Example:

Specification of a server function

```
int func(struct ST *ptr);
```

The function takes *ptr, a pointer to a structure of type ST, as an input and returns a value in the same area.

Definition of the server function

```
int ret = func([INOUT PTR]struct ST *ptr);
```

6.5.4 REF (Passing by Reference)

Description:

This keyword specifies that only the address of the parameter will be passed between the client and the server. REF is useful when the size of the parameter to be passed is large.

This keyword can be used in conjunction with IN and with the data-type keywords PTR, STR, and ARY.

Figure 6.1 illustrates the difference between cases with and without REF.

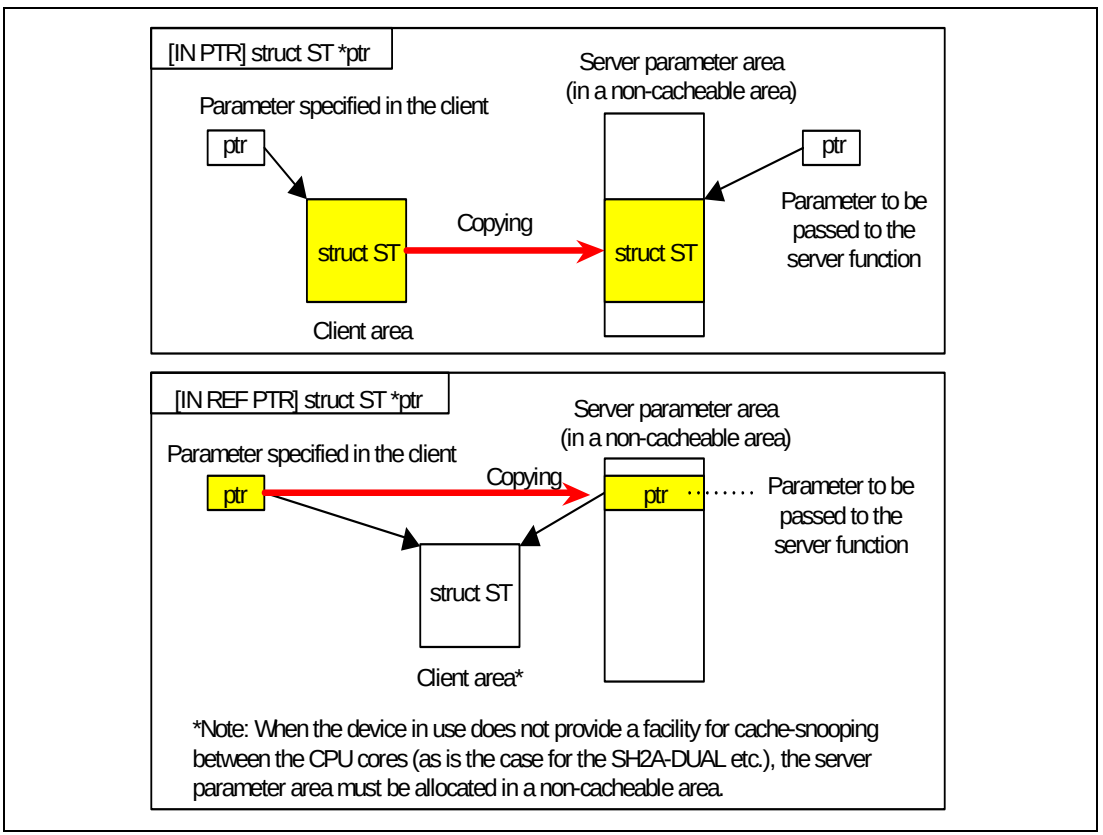

Figure 6.1 REF

Without the REF specification, the data indicated by the pointer variable specified with PTR, STR, or ARY are transferred. The transferred pointer points to a different area from that before the transfer.

With the REF specification, the pointer variable is transferred. In this case, the pointer points to the same area even after the transfer.

Note:

When the device in use does not provide a facility for cache-snooping between the CPU cores (as is the case for the SH2A-DUAL etc.), the server parameter area must be allocated in a non-cacheable area.

Example:

Specification of a server function

```
int func(struct ST *ptr);
```

*ptr is passed to func as the input. ptr is sure to keep pointing to a non-cacheable area.

Definition of the server function

```
int ret = func([IN REF PTR]struct ST *ptr);
```

6.5.5 DESC (Specified Parameter is not Passed from the Client to the Server)

Description:

This keyword specifies that the parameter will not be passed from the client to the server. This keyword can also be used with IN.

Example:

Specification of a server function

```
int func(void);
```

While the actual server function has no parameter, the client application calls an old API function, func, which does have a parameter (int par).

Definition of the server function

```
int ret = func([IN DESC PTR]int par);
```

6.6 Keywords for Data Types

Select one of the following four available keywords.

- DFLT: Default
- PTR: Pointer
- STR: String
- ARY: Array

When ARY is selected, the parameter for COUNT must also be specified.

6.6.1 DFLT (Default)

Input/Output Keywords Available for Use in Combination with DFLT:

IN

Description:

The parameter itself is transferred between the client and server. DFLT is only specifiable when the parameter has any of the following types.

- Integer (signed short, unsigned short, signed long, unsigned long, signed int, or unsigned int)
- Character (signed char or unsigned char)
- Real number (float or double)
- Structure-type or union-type object that is not an array

Example:

Specification of a server function

```
int func(int par);
```

The function takes par as an input and returns a value of type int.

Definition of the server function

```
int ret = func([IN DFLT]int par);
```

6.6.2 STR (String)

Input/Output Keywords Available in Combination with STR:

IN

Description:

The string specified as the parameter is transferred from the client to the server. STR is only specifiable when the parameter is of the character type.

When STR is used with REF, the pointer to the string is transferred instead of the string itself. For details, refer to section 6.5.4, REF (Passing by Reference).

Example:

Specification of a server function

```
int func(const char *s);
```

The function takes string s as an input and returns a value of type int.

Definition of the server function

```
int ret = func([IN STR]const char *s);
```

6.6.3 PTR (Pointer)

Input/Output Keywords Available in Combination with PTR:

All

Description:

The data pointed to by a specified pointer are transferred between the client and the server. PTR is only specifiable when the parameter is of the pointer type.

When IN has been selected and REF is used with PTR, the pointer value is transferred instead of the data it indicates. For details, refer to section 6.5.4, REF (Passing by Reference).

Example:

Specification of a server function

```
void func(struct ST *ptr);
```

The function has no input and returns the structure of type ST that is indicated by *ptr.

Definition of the server function

```
func([OUT PTR]struct ST *ptr);
```

6.6.4 ARY (Array)

Input/Output Keywords Available in Combination with ARY:

All

Description:

The data of a specified array are transferred between the client and the server. ARY is only specifiable when the parameter is of the array type. As the parameter specification that follows this keyword, define a parameter that expresses the first address of the array.

When ARY is used, it must be followed by a definition of COUNT, which indicates the number of elements in the array to be passed between the client and the server. For details, refer to section 6.6.5, COUNT (Number of Elements in an Array).

When IN has been selected and REF is used with ARY, the pointer to the array is transferred instead of the array it indicates. For details, refer to section 6.5.4, REF (Passing by Reference).

Example:

Refer to section 6.6.5, COUNT (Number of Elements in an Array).

6.6.5 COUNT (Number of Elements in an Array)

Description:

When ARY is used, it must be followed by a definition of COUNT, which indicates the number of elements in the array to be passed between the client and the server.

(1) When the input/output keyword selected for ARY is IN

In this case, a parameter specification that includes ARY must be followed by a single definition of COUNT to indicate the number of elements in the array.

```
<parameter including ARY>,<definition of COUNT>
```

COUNT is an expression that specifies the number of elements in the array to be passed from the client to the server.

RPCGEN creates client-stub code that makes an RPC call after setting the size of an input IOVEC structure by using an expression of the kind specified in figure 6.2. This must be taken into account when COUNT is defined. For example, the expression for COUNT must not contain any variables that hold the return value of a function. Figures 6.2 and 6.3 show the code for example 5 (without SVR_AUTH) of the examples introduced following the figures.

```
int func (struct INF * inf, struct ST * ptr )
{
  IOVEC __input[2];
  ...
  __input[ __ulInputParamCount ].pBaseAddress = (UINT8 *)(inf);
  __input[ __ulInputParamCount++ ].ulSize     = sizeof(*inf);
  ...
  if (((UINT32)(inf->count)) > 0UL)
  {
      __input[ __ulInputParamCount ].pBaseAddress = (UINT8 *)(ptr);
      __input[ __ulInputParamCount++ ].ulSize     = sizeof(*ptr) * ((UINT32)(inf->count));
  }
  ...
  rpc_call(...);
}
```

Figure 6.2 Code Output to the Client Stub (IN) for COUNT

```
UINT32 rpcsvr_SMPL_func( rpc_server_stub_info * __pInfo )
{
  ...
  inf = (struct INF *)( __pInfo->pucParamArea + __ulInputParamOffset);
  __ulInputParamOffset += ALIGNUP4(sizeof(*inf));
  if (((UINT32)(inf->count)) > 0UL)
  {
      ptr = (struct ST *)( __pInfo->pucParamArea + __ulInputParamOffset);
  }
  ...
  ret = func( inf, ptr );
  ...
  return ((UINT32)ret);
}
```

Figure 6.3 Code Output to the Server Stub (IN) for COUNT

Examples:

Example 1

Specification of a server function

```
int func(struct ST *ptr);
```

Pointer ptr to an array of 10 elements of the structure type (ST) will be passed.

Definition of the server function

```
int ret = func([IN ARY]struct ST *ptr, [COUNT]10);
```

Example 2

Specification of a server function

```
int func(struct ST *ptr, int count);
```

Pointer ptr to an array of count elements of the structure type (ST) will be passed.

Definition of the server function

```
int ret = func([IN ARY]struct ST *ptr, [COUNT]count,[IN DFLT]int
count);
```

Example 3

Specification of a server function

```
int func(int count , struct ST *ptr);
```

Pointer ptr to an array of count elements of the structure type (ST) will be passed.

Definition of the server function

```
int ret = func([IN DFLT]int count, [IN ARY]struct ST *ptr,
[COUNT]count);
```

Example 4

Specification of a server function

```
int func(struct ST *ptr, int *p_count);
```

Pointer ptr to an array of *p_count elements of the structure type (ST) will be passed.

Definition of the server function

```
int ret = func([IN ARY]struct ST *ptr, [COUNT]*p_count, [IN PTR]int
*p_count);
```

Example 5

Specification of a server function

```
int func(struct INF *inf, struct ST *ptr);
```

ptr points to an array to be given to the server function and inf indicates other input information. It has a type-int member count, which indicates the number of elements in the array pointed to by ptr.

Definition of the server function

```
int ret = func([IN PTR]struct INF *inf, [IN ARY]struct ST *ptr,
[COUNT]inf->count);
```

(2) When the input/output keyword selected for ARY is OUT

In this case, a parameter specification that includes ARY must be followed by two definitions of COUNT to indicate the number of elements in the array.

```
<parameter including ARY>,<definition of first COUNT>,
<definition of second COUNT>
```

The first COUNT is the COUNT keyword followed by an expression that specifies the number of elements in the array for storage of output by the client.

The second COUNT is the COUNT keyword followed by an expression that specifies the number of elements in the array to be actually output by the server. This value must be less than or equal to that specified for the first COUNT.

RPCGEN creates client-stub code that makes an RPC call after setting the size of an input IOVEC structure by using an expression of the kind specified in figure 6.4. RPCGEN also creates server-stub code that makes a server-function call before setting the size of an output IOVEC structure by using an expression of the kind specified in figure 6.5. This must be taken into account in defining the first and second COUNT. For example, an expression for the first COUNT must not contain any variables that hold the return value of a function. Figures 6.4 and 6.5 show the code for example 8 (without SVR_AUTH) of the examples introduced following the figures.

```
  int func (struct INF * inf, struct ST * ptr )
  {
    IOVEC __input[1];
    IOVEC __output[1];
    ...
    __input[ __ulInputParamCount ].pBaseAddress = (UINT8 *)(inf);
    __input[ __ulInputParamCount++ ].ulSize     = sizeof(*inf);

    if (((UINT32)(inf->count)) > 0UL)
    {
       __output[ __ulOutputParamCount ].pBaseAddress   = (UINT8 *)(ptr);
       __output[ __ulOutputParamCount++ ].ulSize    = sizeof(*ptr) * ((UINT32)(inf->count));
    }
    ...
    rpc_call(...);
}
```

Figure 6.4 Code Output to the Client Stub (OUT) for First COUNT

```
UINT32 rpcsvr_SMPL_func( rpc_server_stub_info * __pInfo )
{
  ...
  inf = (struct INF *)( __pInfo->pucParamArea + __ulInputParamOffset);
  __ulInputParamOffset += ALIGNUP4(sizeof(*inf));

  if (((UINT32)(inf->count)) > 0UL)
  {
      ptr = (struct ST *)( __pInfo->pucParamArea + __ulInputParamOffset);
  }

  ret = func( inf, ptr );

  __pInfo->pOutputIOVectorTable[ __ulOutputParamCount   ].pBaseAddress = ptr;
  if (((UINT32)(ret )) > 0UL)
  {
      __pInfo->pOutputIOVectorTable[ __ulOutputParamCount++ ].ulSize = sizeof(*ptr) *
((UINT32)(ret ));
  }
  else
  {
      __pInfo->pOutputIOVectorTable[ __ulOutputParamCount++ ].ulSize = 0UL;
  }
  __pInfo->ulOutputIOVectorTableSize = 1UL;

  return ((UINT32)ret);
}
```

Figure 6.5 Code Output to the Server Stub (OUT) for Second COUNT

Examples:

Example 1

Specification of a server function

```
int func(struct ST *ptr);
```

ptr points to an array for output that has 10 elements. func outputs all 10 elements.

Definition of the server function

```
int ret = func([OUT ARY]struct ST *ptr,[COUNT]10, [COUNT]10));
```

Example 2

Specification of a server function

```
int func(struct ST *ptr, int *p_count);
```

ptr points to an array for output that has 10 elements. *p_count, which is the return parameter of func, indicates the number of elements to be output.

Definition of the server function

```
int ret = func([OUT ARY]struct ST *ptr,
[COUNT]10,[COUNT]*p_count,[OUT PTR]int *p_count);
```

Example 3

Specification of a server function

```
int func(struct ST *ptr);
```

ptr points to an array for output that has 10 elements. The return value from func indicates the number of elements to be output.

Definition of the server function

```
int ret = func([OUT ARY]struct ST *ptr,[COUNT]10, [COUNT]ret);
```

Example 4

Specification of a server function

```
int func(struct ST *ptr, int *p_count);
```

ptr points to an array for output that has *p_count elements. func outputs the return parameter *p_count as the number of elements.

Definition of the server function

```
int ret = func([OUT ARY]struct ST *ptr,
[COUNT]*p_count, [COUNT]*p_count,[INOUT PTR]int *p_count);
```

Example 5

Specification of a server function

```
int func(struct ST *ptr, int count);
```

ptr points to an array for output that has count elements. func outputs all count elements.

Definition of the server function

```
int ret = func([OUT ARY]struct ST *ptr,
[COUNT]count, [COUNT]count,[IN DFLT]int count);
```

Example 6

Specification of a server function

```
int func(struct INF *inf, struct ST *ptr, int *p_count);
```

ptr points to an array for output and inf indicates other input information. It has a type-int member count, which indicates the number of elements in the array pointed to by ptr. func outputs the return parameter *p_count as the number of elements.

Definition of the server function

```
int ret = func([IN PTR]struct INF *inf, [OUT ARY]struct ST *ptr,
[COUNT]inf->count, [COUNT]*p_count, [OUT PTR]int *p_count);
```

Example 7

Specification of a server function

```
int func(struct INF *inf, struct ST *ptr);
```

ptr points to an array for output and inf indicates other input information. It has a type-int member count, which indicates the number of elements in the array pointed to by ptr. func outputs all inf -> count elements.

Definition of the server function

```
int ret = func([IN PTR]struct INF *inf, [OUT ARY]struct ST *ptr,
[COUNT]inf->count, [COUNT]inf->count );
```

Example 8

Specification of a server function

```
int func(struct INF *inf, struct ST *ptr);
```

ptr points to an array for output and inf indicates other input information, including a type-int member count, which indicates the number of elements in the array pointed to by ptr. The return value indicates the number of elements to be output.

Definition of the server function

```
int ret = func([IN PTR]struct INF *inf, [OUT ARY]struct ST *ptr,
[COUNT]inf->count, [COUNT]ret );
```

(3) When the input/output keyword selected for ARY is INOUT

In this case, a parameter specification that includes ARY must be followed by two definitions of COUNT to indicate the number of elements in the array.

```
<parameter including ARY>,<definition of first COUNT>,
<definition of second COUNT>
```

The first COUNT is the COUNT keyword followed by an expression that specifies the number of elements in the array to be passed from the client to the server.

The second COUNT is the COUNT keyword followed by an expression that specifies the number of elements in the array to be actually output by the server. This value must be less than or equal to that specified for the first COUNT.

RPCGEN creates client-stub code that makes an RPC call after setting the size of an input IOVEC structure by using an expression of the kind specified in figure 6.6. RPCGEN also creates server-stub code that makes a server-function call before setting the size of an output IOVEC structure by using an expression of the kind specified in figure 6.7. This must be taken into account in defining the first and second COUNT. For example, an expression for the first COUNT must not contain any variables that hold the return value of a function. Figures 6.6 and 6.7 show the code for example 8 (without SVR_AUTH) of the examples introduced following the figures.

```
int func (struct INF * inf, struct ST * ptr )
{
  IOVEC __input[2];
  IOVEC __output[1];
  ...
  __input[ __ulInputParamCount ].pBaseAddress = (UINT8 *)(inf);
  __input[ __ulInputParamCount++ ].ulSize     = sizeof(*inf);

  if (((UINT32)(inf->count)) > 0UL)
  {
      __input[ __ulInputParamCount ].pBaseAddress = (UINT8 *)(ptr);
      __input[ __ulInputParamCount++ ].ulSize     = sizeof(*ptr) * ((UINT32)(inf->count));
  }

  if (((UINT32)(inf->count)) > 0UL)
  {
      __output[ __ulOutputParamCount ].pBaseAddress   = (UINT8 *)(ptr);
      __output[ __ulOutputParamCount++ ].ulSize   = sizeof(*ptr) * ((UINT32)(inf->count));
  }
  ...
  rpc_call(...);
}
```

Figure 6.6 Code Output to the Client Stub (INOUT) for First COUNT

```
UINT32 rpcsvr_SMPL_func( rpc_server_stub_info * __pInfo )
{
  ...
  inf = (struct INF *)( __pInfo->pucParamArea + __ulInputParamOffset);
  __ulInputParamOffset += ALIGNUP4(sizeof(*inf));

  if (((UINT32)(inf->count)) > 0UL)
  {
      ptr = (struct ST *)( __pInfo->pucParamArea + __ulInputParamOffset);
      __ulInputParamOffset += ALIGNUP4(sizeof(*ptr) * ((UINT32)(inf->count)));
  }
  ...
  ret = func( inf, ptr );

  __pInfo->pOutputIOVectorTable[ __ulOutputParamCount   ].pBaseAddress = ptr;
  if (((UINT32)(ret )) > 0UL)
  {
     __pInfo->pOutputIOVectorTable[ __ulOutputParamCount++ ].ulSize = sizeof(*ptr) *
((UINT32)(ret ));
  }
  else
  {
     __pInfo->pOutputIOVectorTable[ __ulOutputParamCount++ ].ulSize = 0UL;
  }
  __pInfo->ulOutputIOVectorTableSize = 1UL;

  return ((UINT32)ret);
}
```

Figure 6.7 Code Output to the Server Stub (INOUT) for Second COUNT

Examples:

Example 1

Specification of a server function

```
int func(struct ST *ptr);
```

ptr points to an array for input and output that has 10 elements. func outputs all 10 elements.

Definition of the server function

```
int ret = func([INOUT ARY]struct ST *ptr,[COUNT]10, [COUNT]10);
```

Example 2

Specification of a server function

```
int func(struct ST *ptr, int *p_count);
```

ptr points to an array for input and output that has 10 elements. *p_count, which is the return parameter of func, indicates the number of elements to be output.

Definition of the server function

```
int ret = func([INOUT ARY]struct ST *ptr,
[COUNT]10,[COUNT]*p_count,[OUT PTR]int *p_count);
```

Example 3

Specification of a server function

```
int func(struct ST *ptr);
```

ptr points to an array for input and output that has 10 elements. The return value from func indicates the number of elements to be output.

Definition of the server function

```
int ret = func([INOUT ARY]struct ST *ptr,[COUNT]10, [COUNT]ret);
```

Example 4

Specification of a server function

```
int func(struct ST *ptr, int *p_count);
```

ptr points to an array for input and output that has *p_count elements. func outputs the return parameter *p_count as the number of elements.

Definition of the server function

```
int ret = func([INOUT ARY]struct ST *ptr,
[COUNT]*p_count,[COUNT]*p_count,[INOUT PTR]int *p_count);
```

Example 5

Specification of a server function

```
int func(struct ST *ptr, int count);
```

ptr points to an array for input and output that has count elements. func outputs all count elements.

Definition of the server function

```
int ret = func([INOUT ARY]struct ST *ptr,
[COUNT]count, [COUNT]count, [IN DFLT]int count);
```

Example 6

Specification of a server function

```
int func(struct INF *inf, struct ST *ptr, int *p_count);
```

ptr points to an array for input and output and inf indicates other input information, including a type-int member count, which indicates the number of elements in the array pointed to by ptr. func outputs the return parameter *p_count as the number of elements.

Definition of the server function

```
int ret = func([IN PTR]struct INF *inf, [INOUT ARY]struct ST *ptr,
[COUNT]inf->count, [COUNT]*p_count, [OUT PTR]int *p_count);
```

Example 7

Specification of a server function

```
int func(struct INF *inf, struct ST *ptr);
```

ptr points to an array for input and output and inf indicates other input information, including a type-int member count, which indicates the number of elements in the array pointed to by ptr. func outputs all inf -> count elements.

Definition of the server function

```
int ret = func([IN PTR]struct INF *inf, [INOUT ARY]struct ST *ptr,
[COUNT]inf->count, [COUNT]inf->count );
```

Example 8

Specification of a server function

```
int func(struct INF *inf, struct ST *ptr);
```

ptr points to an array for input and output and inf indicates other input information, including a type-int member count, which indicates the number of elements in the array pointed to by ptr. The return value indicates the number of elements to be output.

Definition of the server function

```
int ret = func([IN PTR]struct INF *inf, [INOUT ARY]struct ST *ptr,
[COUNT]inf->count, [COUNT]ret );
```

6.7 Optional Keywords

The following optional keywords are available to control output to the client or server stub for the individual server functions.

- SVRSTUB: Server stub
- SVRFUNC: Server function name
- CLNTSTUB: Client stub
- UNACK: Asynchronous call
- CLNTCOPYCBK: RPC call by rpc_call_copycbk()

Two or more optional keywords may be specifiable. Note, however, that the same keyword must not be used more than once. When a combination of keywords specified by the user is not valid, RPCGEN shows an error message and terminates processing.

Multiple keywords must be delimited by commas.

Table 6.2 shows the available combinations of optional keywords.

Table 6.2 Combinations of Optional Keywords

	SVRSTUB	SVRFUNC	CLNTSTUB	UNACK	CLNTCOPYCBK
SVRSTUB		—	✓	✓	✓
SVRFUNC			✓	✓	✓
CLNTSTUB				—	—
UNACK					✓
CLNTCOPYCBK					

✓: Available

—: Not available (RPCGEN reports an error and terminates processing)

6.7.1 SVRSTUB (Server Stub)

Format:

```
[SVRSTUB] <server-stub function name>
```

Description:

A server stub provided by the user is to be used, so RPCGEN does not create a server stub.

Example:

Specification of a server function

```
int func(int par);
```

The function takes par as an input and returns a value of type int. RPCGEN will not create a server stub for func (i.e. will not create func_svrstub) because the user will provide this server stub.

Definition of the server function

```
int ret = func([IN DFLT]int par)[SVRSTUB]func_svrstub;
```

6.7.2 SVRFUNC (Server Function Name)

Format:

```
[SVRFUNC]<replacement server function name>
```

Description:

RPCGEN replaces the name of the server function to be called by the server stub with a specific name. Use SVRFUNC when the name of a function as called by the client does not match the name of the function as called by the server.

SVRFUNC cannot be used with SVRSTUB.

Example:

Specification of a server function

```
int func(int par);
```

The function takes par as an input and returns a value of type int. func on the server, however, is named func_main.

Definition of the server function

```
int ret = func([IN DFLT]int par)[SVRFUNC]func_main;
```

6.7.3 CLNTSTUB (Client Stub)

Format:

```
[CLNTSTUB]
```

Description:

A client stub provided by the user is to be used, so RPCGEN does not create a client stub.

Example:

Specification of a server function

```
int func(int par);
```

The function takes par as an input and returns a value of type int. RPCGEN will not create a client stub for func because the user will provide this client stub.

Definition of the server function

```
int ret = func([IN DFLT]int par)[CLNTSTUB];
```

6.7.4 UNACK (Asynchronous Call)

Format:

```
[UNACK]
```

Description:

The client makes RPC calls in asynchronous mode. When UNACK is not used, RPC calls are made in synchronous mode.

UNACK cannot be used with CLNTSTUB.

Note:

When UNACK is used, the return value of the server function must be of type void and the input/output keyword for parameters must be IN.

Example:

Specification of a server function

```
void func(int par);
```

The function takes par as an input. The client makes RPC calls in asynchronous mode.

Definition of the server function

```
func([IN DFLT]int par)[UNACK];
```

6.7.5 CLNTCOPYCBK (RPC Call by rpc_call_copycbk())

Format:

```
[CLNTCOPYCBK]<function 1> <function 2>
```

Description:

RPC calls are started by rpc_call_copycbk(). <function 1> and <function 2> are copy callback functions specified in rpc_call_copycbk(). Specification of the two callback functions is mandatory.

The prototype for the callback functions must conform with the RPC specifications and has to be declared in a file included by the client stub.

Example:

Specification of a server function

```
int func(struct ST *ptr);
```

ptr is an array for output and has 10 elements, all of which are output by func. copy1() is used for transfer from the client to the server while copy2() is used for transfer from the server to the client.

Definition of the server function

```
int ret = func([OUT ARY]struct ST *ptr,[COUNT]10,
[COUNT]10)[CLNTCOPYCBK] copy1 copy2;
```

Section 7 Server Functions Not Supported by RPCGEN

This section gives the types of functions not supported by RPCGEN. If a server function you are using falls under the definitions in this section, use SVRSTUB or CLNTSTUB so that RPCGEN will not create a server or client stub for the server function. A substitute server or client stub must be implemented by the user.

7.1 Parameter

RPCGEN is not capable of correctly creating a client or server stub for server functions that have parameters of the following types.

(1) Parameters with data types of undefined size

RPCGEN cannot handle parameters with data types that have undefined size. When such a parameter is a pointer, however, using REF allows the creation of stub code for passing of the pointer itself.

Example:

Specification of a server function

```
int func(void *par);
```

The function takes ptr, a pointer to data of undefined size, as an input and returns an int-type value.

Definition of the server function

```
int ret = func([IN PTR]void *par);
```

In this case, an error will occur in compilation of the client stub.

```
int ret = func([IN REF PTR]void *par);
```

In this case, code to pass the value of par to the server will be created.

(2) Multiple-level pointers

RPCGEN only recognizes the first level of multiple-level pointers.

Example:

Specification of a server function

```
int func(int **ptr);
```

The function takes ptr, a pointer to a pointer to int-type data, as an input.

Definition of the server function

```
int ret = func([IN PTR] int **ptr);
```

In this case, RPCGEN creates code to transfer *ptr from the client to the server. *ptr is a pointer to the target int-type data, but this data (**ptr) will not be transferred.

(3) Function pointer

RPCGEN does not support server functions whose parameters are pointers to functions.

7.2 Return Value

(1) Pointer

If a function returns a pointer-type value, this return value indicates an address in the server. RPCGEN does not create any code to transfer the data indicated by the pointer from the server to the client.

Example:

Specification of a server function

```
int *func(int par);
```

The return value is a pointer to int-type data.

Definition of the server function

```
int *ret = func([IN DFLT]int par);
```

In this case, ret is a pointer to int-type data returned within the server by the server function.

Section 8 Application Interface

8.1 Client-Stub Functions Generated by RPCGEN

The client-stub functions are called by the client application.

This section gives the specifications for the client-stub functions generated by RPCGEN. The client-stub functions must be called in accord with the definitions in these specifications. If CLNTSTUB is used to prevent creation of client-stub functions, the user must implement client-stub functions in accord with the specifications.

For details on SVR_AUTH, refer to section 5.5.8, SVR_AUTH.

(1) When SVR_AUTH has not been specified

Each of the client-stub functions has the same API as the actual server function. RPCGEN outputs the client-stub functions themselves to <config file>_clnt.c but does not output prototype declarations. Prepare a header file with the required prototype declarations and specify this header file in the definition of CLNT_INCFILE or GLOBAL_INCFILE.

Example:

Contents of a config file

```
SVR_NAME{ EXAMPLE };
SVR_ID{ 1 };
SVR_VERSION{ 2 };
// SVR_AUTH;
RPC_FUNC{
    int ret = func1([IN DFLT]int par1, [IN DFLT]int par2);
    func2([OUT PTR]struct ST *ptr);
};
```

Specification of a client-stub function

```
int func1(int par1, int par2);
void func2(struct ST *ptr);
```

(2) When SVR_AUTH has been specified

Each of the client-stub functions has a different API from the corresponding actual server function and the server ID and server version information are added as parameters. RPCGEN outputs the client-stub functions themselves to <config file>_clnt.c and the prototype declarations to <config file>_clnt.h.

The name of each client-stub function takes the form rpcclnt_<server name>_<server function name>. <Server name> refers to the server name defined with SVR_NAME and <server function name> refers to server function names defined between the braces {} of RPC_FUNC.

The first and second parameters of the client-stub functions are the server ID and server version, respectively. The names of the parameters are fixed to __ulID and __ulVers. The third and subsequent parameters are the first and subsequent parameters of the actual server function and their names are those defined in RPC_FUNC.

Example:

Contents of a config file

```
SVR_NAME{ EXAMPLE };
SVR_ID{ 1 };
SVR_VERSION{ 2 };
SVR_AUTH;
RPC_FUNC{
    int ret = func1([IN DFLT]int par1, [IN DFLT]int par2);
    func2([OUT PTR]struct ST *ptr);
};
```

Specification of a client-stub function

```
int rpcclnt_EXAMPLE_func1(UINT32 __ulID, UINT32 __ulVers, int par1, int
par2);
void rpcclnt_EXAMPLE_func2(UINT32 __ulID, UINT32 __ulVers, struct ST
*ptr);
```

8.2 Server-Stub Functions Generated by RPCGEN

RPCGEN outputs the server-stub functions themselves to <config file>_svr.c and the prototype declarations to <config file>_svr.h.

The name of each server-stub function takes the form rpcsvr_<server name>_<server function name>. <Server name> refers to the server name defined with SVR_NAME and <server function name> refers to server function names defined between the braces {} of RPC_FUNC.

Since the server-stub functions are called from within the RPC library, the user usually does not need to recognize the APIs of the server-stub functions output by RPCGEN. If SVRSTUB is used to prevent creation of the server-stub functions, however, the user must implement the server-stub functions in accord with the specifications.

Example:

Contents of a config file

```
SVR_NAME{ EXAMPLE };
RPC_FUNC{
    int ret = func1([IN DFLT]int par);
    func([OUT PTR]struct ST *ptr);
};
```

Specification of a client-stub function

```
UINT32 rpcsvr_EXAMPLE_func1(rpc_server_stub_info * const __pInfo);
UINT32 rpcsvr_EXAMPLE_func2(rpc_server_stub_info * const __pInfo);
```

8.3 Server-Initialization Function

The server-initialization function is called by the server application and starts up the server by using rpc_start_server() or rpc_start_server_with_paramarea().

When SVR_NOINIT has not been specified, RPCGEN generates a server-initialization function rpcsvr_<server name>_init. RPCGEN outputs the server-initialization function itself to <config file>_svr.c and the prototype declaration to <config file>_svr.h.

<Server name> refers to the server name defined with SVR_NAME.

When SVR_NOINIT has been specified, RPCGEN does not generate a server-initialization function. The server-initialization function must be provided by the user if one is required.

The specification of the server-initialization function generated by RPCGEN is as follows.

- C-language API

```
INT32 rpcsvr_<server name>_init( rpc_svr_config *__config);
```

- Return value

  The return value of rpc_start_server() or rpc_start_server_with_paramarea()
- Parameter: __config

  The definition of an rpc_svr_config structure is given below. Each of the members has the same meaning as that of the rpc_server_info structure.

```
typedef struct {
    UINT32 ulRPCServerID;
    UINT32 ulRPCServerVersion;
    UINT32 ServerStubTaskPriority;
    UINT32 ulStubStackSize;
    UINT32 ulMaxParamAreaSize;
    void   *pUserDefinedData;
} rpc_svr_config;
```

- Specification

  The server is started up by calling rpc_start_server() when SVR_STATIC has not been specified or rpc_start_server_with_paramarea() when SVR_STATIC has been specified.

  When SVR_AUTH has not been specified, however, the definitions of server ID and version in __config. UlRPCServerID and __config. UlRPCServerVersion are ignored. Instead, the macros RPCSVR_ID_<server name> and RPCSVR_VERS_<server name> output to <config file>_public.h are taken as the server ID and version in starting up the server.

8.4 Server-Stub Function Table

The server-stub function table is rpc_server_info.ServerStubList, the table defined by rpc_start_server() or rpc_start_server_with_paramarea() called by the server-initialization function.

(1) With neither SVR_NOSTUBTBL nor SVR_NOINIT

RPCGEN generates the server-initialization function and server-stub function table. Only the server-initialization function generated by RPCGEN looks up the server-stub function table. Thus RPCGEN outputs the server-stub function table as static data to <config file>_svr.c.

(2) Without SVR_NOSTUBTBL and with SVR_NOINIT

RPCGEN does not generate the server-initialization function but does generate the server-stub function table. RPCGEN outputs the server-stub function table in the form given below to <config file>_svr.c. The server-initialization function provided in the application must look up the server-stub function table to call rpc_start_server() or rpc_start_server_with_paramarea().

```
UINT32 (* const __rpcsvr_<server name>
_StubTable[])(rpc_server_stub_info *) =
{
<server-stub function name>,
<server-stub function name>,
};
```

<Server name> refers to the server name defined with SVR_NAME.

(3) With SVR_NOSTUBTBL and without SVR_NOINIT

RPCGEN does not generate the server-stub function table but does generate the server-initialization function. RPCGEN assumes that the server-stub function table shown below is provided in the application and outputs the server-initialization function, which externally refers to the table, to <config file>_svr.c.

```
UINT32 (* const __rpcsvr_<server name>_StubTable[])
(rpc_server_stub_info *) =
{
<server-stub function name>,
<server-stub function name>,
};
```

<Server name> refers to the server name defined with SVR_NAME.

(4) With SVR_NOSTUBTBL and SVR_NOINIT

RPCGEN generates neither a server-initialization function nor a server-stub function table, so they must be provided in the application.

8.5 Server-Shutdown Function

The server-shutdown function is called by the server application and stops the server by using rpc_stop_server().

When SVR_NOSHUTDOWN has not been specified, RPCGEN generates a server-shutdown function rpcsvr_<server name>_shutdown. RPCGEN outputs the server-shutdown function itself to <config file>_svr.c and the prototype declaration to <config file>_svr.h.

<Server name> refers to the server name defined with SVR_NAME.

When SVR_NOSHUTDOWN has been specified, RPCGEN does not generate a server-shutdown function. The server-shutdown function must be provided by the user if one is required.

The specification of the server-shutdown function generated by RPCGEN is as follows.

- C-language API

```
INT32 rpcsvr_<server name>_shutdown(UINT32 __ulServerID, UINT32
__ulServerVersion);
```

- Return value

  The return value of rpc_stop_server()
- Parameter: __ulServerID and __ulServerVersion

  These parameters indicate the server ID and server version, respectively. When SVR_AUTH has not been specified, however, the server ID and server version are ignored. Instead, the macros RPCSVR_ID_<server name> and RPCSVR_VERS_<server name> output to <config file>_public.h are taken as the server ID and server version in stopping the server.
- Specification

  The server is stopped by calling rpc_stop_server() and no callback function is specifiable for the server-shutdown function. If you wish to use a callback function, specify SVR_NOSHUTDOWN and implement the server-shutdown function on the user side.

8.6 Client-Initialization Function

The client-initialization function is called by the client application and initiates connection to the server by using rpc_connect().

When CLNT_NOINIT has not been specified, RPCGEN generates a client-initialization function rpcclnt_<server name>_init. RPCGEN outputs the client-initialization function itself to <config file>_clnt.c and the prototype declaration to <config file>_clnt.h.

<Server name> refers to the server name defined with SVR_NAME.

When CLNT_NOINIT has been specified, RPCGEN does not generate a client-initialization function. The client-initialization function must be provided by the user if one is required.

The specification of the client-initialization function generated by RPCGEN is as follows.

- C-language API

```
INT32 rpcclnt_<server name>_init(UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion);
```

- Return value

  The return value of rpc_connect()
- Parameter: __ulRPCServerID and __ulRPCServerVersion

  These parameters indicate the server ID and server version, respectively. When SVR_AUTH has not been specified, however, the server ID and server version are ignored. Instead, the macros RPCSVR_ID_<server name> and RPCSVR_VERS_<server name> output to <config file>_public.h are taken as the server ID and version in initiating connection to the server.
- Specification

  Connection to the server is initiated by using rpc_connect().

8.7 Client-Shutdown Function

The client-shutdown function is called by the client application and terminates the connection to the server by using rpc_disconnect().

When CLNT_NOSHUTDOWN has not been specified, RPCGEN generates a client-shutdown function rpcclnt_<server name>_shutdown. RPCGEN outputs the client-shutdown function itself to <config file>_clnt.c and the prototype declaration to <config file>_clnt.h.

<Server name> refers to the server name defined with SVR_NAME.

When CLNT_NOSHUTDOWN has been specified, RPCGEN does not generate a client-shutdown function. The client-shutdown function must be provided by the user if one is required.

The specification of the client-shutdown function generated by RPCGEN is as follows.

- C-language API

```
INT32 rpcclnt_<server name>_shutdown(UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion);
```

- Return value

  The return value of rpc_disconnect()
- Parameter: __ulRPCServerID and __ulRPCServerVersion

  These parameters indicate the server ID and server version, respectively. When SVR_AUTH has not been specified, however, the server ID and server version are ignored. Instead, the macros RPCSVR_ID_<server name> and RPCSVR_VERS_<server name> output to <config file>_public.h are taken as the server ID and server version in terminating a connection to the server.
- Specification

  Connection to the server is terminated by using rpc_disconnect() and no callback function is specifiable for the client-shutdown function. In the current HI7200/MP specification, rpc_disconnect() handles no processing other than terminating the connection to the server.

8.8 rpc_retval_adr()

rpc_retval_adr() is a client function to be provided by the user when CLNT_CALLCHK (refer to section 5.6.3, CLNT_CALLCHK) is specified.

rpc_retval_adr() must be created according to the specification given below.

- C-language API

```
INT32 *rpc_retval_adr(void);
```

- Specification

  The address to hold the return value that depends on the context (task or non-task) in which rpc_retval_adr() was called is returned.

- Example of implementation (for the HI7200/MP)

```
#define MAXTSKID  10    /* Maximum task ID */
    static INT32 retval[MAXTSKID+1];  /* Array to hold the return */
                                      /* value in the context */
                                      /* [0]: Non-task context */
                                      /* [1] to [MAXTSKID]: Task ID */
    INT32 *rpc_retval_adr(void)
    {
        INT32 *retval_adr;
        ID      myid;

        if(sns_ctx() == TRUE)
        {
            /* In a non-task context */
            retval_adr = retval;
        }
        else
        {
            /* In a task context */
            get_tid(%myid);
            retval_adr = &retval[GET_LOCALID(myid)];
        }
        return retval_adr;
    }
```

8.9 <Config file>_public.h

The following macro definitions are output to <config file>_public.h.

```
#define RPCSVR_ID_<server name>  <server ID>
#define RPCSVR_VERS_<server name>  <server version>
```

<Server name>, <server ID>, and <server version> refer to the server name, server ID, and server version defined with SVR_NAME, SVR_ID, and SVR_VERSION, respectively.

The position of <config file>_public.h depends on whether SVR_AUTH has been specified or not. For details, refer to section 5.5.8, SVR_AUTH.

8.10 Local Variables Used in the Source Code Created by RPCGEN

In the source code of the client and server stubs, the names of local variables start with two underscores ("__"). In the client and server stubs, on the other hand, RPCGEN uses the parameter names specified in the definitions of server functions. For this reason, the parameter names specified in the definitions of server functions must not start with two underscores ("__").

Section 9 Notes

(1) Check that the source program generated by RPCGEN works as intended.

We do not guarantee the operation of the program generated by RPCGEN. Be sure to check that the program works as you intended.

RPCGEN does not report errors except in cases where a fatal error has been detected (i.e. when continuing with current processing is impossible).

(2) Do not modify the contents of the files generated by RPCGEN.

Even if the contents of the source files generated by RPCGEN are not as you expected, do not modify the files manually. This is because files that have been manually modified by the user may be overwritten when RPCGEN is re-executed. Instead of manually correcting the generated files, check and correct the contents of config files.

Section 10 Error Messages

RPCGEN shows an error message and terminates processing only when it has detected a fatal error (i.e. when continuing with the current processing is impossible). In other words, RPCGEN behaves as if it has successfully completed code generation in many cases where it has not in fact done so.

Cases in which no errors will be detected:

Note: "->" in the list below indicates a behavior in the current implementation. The behavior may be changed in a future implementation.

- A specific keyword is used multiple times.
  -> The first instance of the keyword to be found is valid.
- A server function with a specific name is declared multiple times.
  -> An error will be reported on compilation of the generated stub.
- An invalid keyword is specified.
  -> This is simply ignored.
- SVR_ID (server ID) is a value with size exceeding 4 bytes.
  -> An error will be reported on compilation of the file to include rpc_public.h.
- A parameter for which PTR has been specified is not of the pointer type.
  -> RPC calls will lead to abnormal operation such as CPU exceptions.

10.1 Format

Error messages are output in the format shown below. The following sections list the error messages and their meanings.

```
<Config file> : (E) <Error message>
```

10.2 General

`Cannot open configuration file.`

RPCGEN cannot access the specified config file.

`No <keyword> definition.`

<keyword> must not be omitted.

`The <keyword> directory doesn't exist.`

The directory <keyword> was specified as a path by a definition such as PUB_INCPATH but does not exist.

`Cannot create file. File: <filename>`

RPCGEN cannot generate the file <filename>.

`Cannot create temporary file.`

RPCGEN cannot generate intermediate files in the current directory.

10.3 Definition Errors (other than RPC_FUNC)

`Illegal <keyword> definition.`

The definition of parameter <keyword> is illegal.

`Cannot define both <keyword 1> and <keyword 2>.`

<keyword 1> and <keyword 2> must not be specified at the same time.

10.4 Definition Errors (RPC_FUNC)

`[RPC_FUNC] Cannot define both <keyword 1> and <keyword 2> for "<function name>".`

In the definition of <function name>, <keyword 1> and <keyword 2> must not be specified at the same time.

`[RPC_FUNC] Illegal <keyword> definition for "<function name>".`

In the definition of <function name>, parameter <keyword> is illegal.

`[RPC_FUNC] Illegal COUNT definition for "<function name>".`

In the definition of <function name>, COUNT following ARY is illegal.

Section 11 Samples

This section introduces samples of the code in a config file and the corresponding output files. To create these samples, we have used the C standard library functions and the HI7200/MP service calls listed in table 11.1. This is simply because we have assumed that most users will understand these functions, not because the functions should necessarily be distributed by the RPC facility.

Table 11.1 Files Provided for Illustration

Server Function	Description
double atof(const char *nptr);	C standard library function (stdlib.h)
int atoi(const char *nptr);	C standard library function (stdlib.h) with REF
ER ref_tsk2(ID tskid, T_RTSK *pk_rtsk);	Service call ref_tsk
ER ref_sem2(ID semid, T_RSEM *pk_rsem);	Service call ref_sem

11.1 Config File (sample.x)

```
/*********************************************************************
  File Header Comment : Sample Config File for RPCGEN
*********************************************************************/

PUB_INCPATH     {"include"};    // Output path for xxx_public.h
SVR_SOURCEPATH  {"server"};     // Output path xxx_svr.c
SVR_INCPATH     {"server"};     // Output path xxx_svr.h
CLNT_SOURCEPATH {"client"};     // Output path xxx_clnt.c
CLNT_INCPATH    {"client"};     // Output path xxx_clnt.h and
xxx_private.h

GLOBAL_INCFILE  {<stdlib.h> "kernel.h"};
                        // Include files for xxx_svr.c and xxx_clnt.c
//SVR_INCFILE   {"..." "..."};  // Include files for xxx_svr.c
//CLNT_INCFILE  {"..." "..."};  // Include files for xxx_clnt.c

SVR_NAME    {SMPL};             // Server name
SVR_ID      {1};                // Server ID
SVR_VERSION {2};                // Server version
// SVR_AUTH;                    // How to authenticate

// SVR_NOINIT;             // Don't generate server initialize function
// SVR_NOSTUBTBL;          // Don't generate server stub function table
// SVR_NOSHUTDOWN;          // Don't generate server shutdown function
// SVR_STATIC {size, section};  // Use static server
SVR_SECTION {C_SAMPLE};         // Section name for server stubs

// CLNT_NOINIT;            // Don't generate client initialize function
// CLNT_NOSHUTDOWN;        // Don't generate server shutdown function
CLNT_CALLCHK;                   // Save return value of RPC-call
CLNT_SECTION{C_SAMPLE};         // Section name for client stubs

RPC_FUNC                        // Define server functions
```

```
{
    // stdlib.h
    [RETEXT] double ret = atof ([IN STR] const char *nptr);
                    // nptr does not have to point to non-cached area
    int ret = atoi ([IN REF STR] const char *nptr);
                    // nptr must point to non-cached area

    // HI7200/MP service calls (renamed from original service call
name)
    ER ercd = ref_tsk2 ([IN DFLT] ID tskid, [OUT PTR] T_RTSK *pk_rtsk);
    ER ercd = ref_sem2 ([IN DFLT] ID semid, [OUT PTR] T_RSEM *pk_rsem);
};
```

11.2 sample_clnt.h

```
/************************************************************************
  File Header Comment : Sample Config File for RPCGEN
************************************************************************/

/************************************************************************
 * Do not edit, unless you really know what you are doing,
 * this file was automatically generated by:
 *
 *    rpcgen.pl on 2007/11/28  12:54
 *
 * Be aware that this file may be generated as part of the build and
 * therefore any changes made by hand will possibly be lost on a
 * rebuild.
************************************************************************/

#ifndef _RPC_SMPL_CLNT_H
#define _RPC_SMPL_CLNT_H

#ifdef __cplusplus
extern "C" {
#endif
INT32 rpcclnt_SMPL_init ( UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion );
INT32 rpcclnt_SMPL_shutdown ( UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion );
double atof (  const char * nptr );
int atoi (  const char * nptr );
ER ref_tsk2 ( ID tskid,  T_RTSK * pk_rtsk );
ER ref_sem2 ( ID semid,  T_RSEM * pk_rsem );

extern INT32 *rpc_retval_adr( void );

#ifdef __cplusplus
}
```

```
#endif
#endif /* End of _RPC_SMPL_CLNT_H */
```

11.3 sample_private.h

```
/************************************************************************
  File Header Comment : Sample Config File for RPCGEN
************************************************************************/

/************************************************************************
 * Do not edit, unless you really know what you are doing,
 * this file was automatically generated by:
 *
 *    rpcgen.pl on 2007/11/28  12:54
 *
 * Be aware that this file may be generated as part of the build and
 * therefore any changes made by hand will possibly be lost on a
 * rebuild.
************************************************************************/

#ifndef _RPC_SMPL_PRIVATE_H
#define _RPC_SMPL_PRIVATE_H


enum
{
    RPC_SMPL_ATOF,
    RPC_SMPL_ATOI,
    RPC_SMPL_REF_TSK2,
    RPC_SMPL_REF_SEM2,
};

#endif /* End of _RPC_SMPL_PRIVATE_H */
```

11.4 sample_clnt.c

```
/*********************************************************************
  File Header Comment : Sample Config File for RPCGEN
*********************************************************************/

/*********************************************************************
 * Do not edit, unless you really know what you are doing,
 * this file was automatically generated by:
 *
 *    rpcgen.pl on 2007/11/28  12:54
 *
 * Be aware that this file may be generated as part of the build and
 * therefore any changes made by hand will possibly be lost on a
 * rebuild.
*********************************************************************/

#include <string.h>
#include "types.h"
#include "rpc_public.h"
#include <stdlib.h>
#include "kernel.h"
#include "sample_public.h"
#include "sample_clnt.h"
#include "sample_private.h"


#pragma section C_SAMPLE

double atof (  const char * nptr )
{
    UINT32 __ulLastOutputIOVectorSize;
    rpc_call_info __info;
    UINT32 __ulInputParamCount = 0UL;
    IOVEC __input[1];
```

```
    UINT32 __ulOutputParamCount = 0UL;
    IOVEC __output[1];
    double ret;
    UINT32 __ulReturn;

   __info.ulMarshallingType          = 0UL;
   __info.ulServerID                 = RPCSVR_ID_SMPL;
   __info.ulServerVersion            = RPCSVR_VERS_SMPL;
   __info.ulServerProcedureID        = RPC_SMPL_ATOF;
   __info.AckMode                    = RPC_ACK;
   __info.pInputIOVectorTable        = __input;
   __info.ulInputIOVectorTableSize  = sizeof(__input) / sizeof (IOVEC);
   __info.pOutputIOVectorTable       = __output;
   __info.ulOutputIOVectorTableSize = sizeof(__output) / sizeof
(IOVEC);
   __info.pulLastOutputIOVectorSize   = &__ulLastOutputIOVectorSize;
   __info.pulReturnValue              = (UINT32 *)&__ulReturn;

   __input[ __ulInputParamCount ].pBaseAddress = (UINT8 *)(nptr);
   __input[ __ulInputParamCount++ ].ulSize = strlen((const char
*)(nptr)) + 1UL;

   __output[ __ulOutputParamCount ].pBaseAddress   = &ret;
   __output[ __ulOutputParamCount++ ].ulSize       = sizeof(double);

   *rpc_retval_adr() =     rpc_call( &__info );

   return ret;
}


int atoi (  const char * nptr )
{
   UINT32 __ulLastOutputIOVectorSize;
   rpc_call_info __info;
```

```
    UINT32 __ulInputParamCount = 0UL;
    IOVEC __input[1];
    int ret;

    __info.ulMarshallingType          = 0UL;
    __info.ulServerID                 = RPCSVR_ID_SMPL;
    __info.ulServerVersion            = RPCSVR_VERS_SMPL;
    __info.ulServerProcedureID        = RPC_SMPL_ATOI;
    __info.AckMode                    = RPC_ACK;
    __info.pInputIOVectorTable        = __input;
    __info.ulInputIOVectorTableSize  = sizeof(__input) / sizeof (IOVEC);
    __info.pOutputIOVectorTable        = NULL;
    __info.ulOutputIOVectorTableSize   = 0UL;
    __info.pulLastOutputIOVectorSize   = &__ulLastOutputIOVectorSize;
    __info.pulReturnValue              = (UINT32 *)&ret;

    __input[ __ulInputParamCount ].pBaseAddress = (UINT8 *)&nptr;
    __input[ __ulInputParamCount++ ].ulSize     = sizeof(UINT32 *);

    *rpc_retval_adr() =     rpc_call( &__info );

    return ret;
}


ER ref_tsk2 ( ID tskid,  T_RTSK * pk_rtsk )
{
    UINT32 __ulLastOutputIOVectorSize;
    rpc_call_info __info;
    UINT32 __ulInputParamCount = 0UL;
    IOVEC __input[1];
    UINT32 __ulOutputParamCount = 0UL;
    IOVEC __output[1];
    ER ercd;
```

```
    __info.ulMarshallingType          = 0UL;
    __info.ulServerID                 = RPCSVR_ID_SMPL;
    __info.ulServerVersion            = RPCSVR_VERS_SMPL;
    __info.ulServerProcedureID        = RPC_SMPL_REF_TSK2;
    __info.AckMode                    = RPC_ACK;
    __info.pInputIOVectorTable        = __input;
    __info.ulInputIOVectorTableSize  = sizeof(__input) / sizeof (IOVEC);
    __info.pOutputIOVectorTable       = __output;
    __info.ulOutputIOVectorTableSize  = sizeof(__output) / sizeof
(IOVEC);
    __info.pulLastOutputIOVectorSize    = &__ulLastOutputIOVectorSize;
    __info.pulReturnValue               = (UINT32 *)&ercd;

    __input[ __ulInputParamCount ].pBaseAddress = &tskid;
    __input[ __ulInputParamCount++ ].ulSize     = sizeof(ID);

    __output[ __ulOutputParamCount ].pBaseAddress  = (UINT8 *)(pk_rtsk);
    __output[ __ulOutputParamCount++ ].ulSize      = sizeof(*pk_rtsk);

    *rpc_retval_adr() =     rpc_call( &__info );

    return ercd;
}


ER ref_sem2 ( ID semid,  T_RSEM * pk_rsem )
{
    UINT32 __ulLastOutputIOVectorSize;
    rpc_call_info __info;
    UINT32 __ulInputParamCount = 0UL;
    IOVEC __input[1];
    UINT32 __ulOutputParamCount = 0UL;
    IOVEC __output[1];
    ER ercd;
```

```
    __info.ulMarshallingType          = 0UL;
    __info.ulServerID                 = RPCSVR_ID_SMPL;
    __info.ulServerVersion            = RPCSVR_VERS_SMPL;
    __info.ulServerProcedureID        = RPC_SMPL_REF_SEM2;
    __info.AckMode                    = RPC_ACK;
    __info.pInputIOVectorTable        = __input;
    __info.ulInputIOVectorTableSize  = sizeof(__input) / sizeof (IOVEC);
    __info.pOutputIOVectorTable       = __output;
    __info.ulOutputIOVectorTableSize  = sizeof(__output) / sizeof
(IOVEC);
    __info.pulLastOutputIOVectorSize    = &__ulLastOutputIOVectorSize;
    __info.pulReturnValue               = (UINT32 *)&ercd;

    __input[ __ulInputParamCount ].pBaseAddress = &semid;
    __input[ __ulInputParamCount++ ].ulSize     = sizeof(ID);

    __output[ __ulOutputParamCount ].pBaseAddress  = (UINT8 *)(pk_rsem);
    __output[ __ulOutputParamCount++ ].ulSize      = sizeof(*pk_rsem);

    *rpc_retval_adr() =     rpc_call( &__info );

    return ercd;
}


INT32 rpcclnt_SMPL_init ( UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion )
{
    return rpc_connect( RPCSVR_ID_SMPL, RPCSVR_VERS_SMPL );
}

INT32 rpcclnt_SMPL_shutdown ( UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion )
{
    return rpc_disconnect( RPCSVR_ID_SMPL, RPCSVR_VERS_SMPL, NULL, 0UL
);
```

```
}
```

11.5 sample_svr.h

```
/*********************************************************************
  File Header Comment : Sample Config File for RPCGEN
*********************************************************************/

/*********************************************************************
 * Do not edit, unless you really know what you are doing,
 * this file was automatically generated by:
 *
 *    rpcgen.pl on 2007/11/28  12:54
 *
 * Be aware that this file may be generated as part of the build and
 * therefore any changes made by hand will possibly be lost on a
 * rebuild.
*********************************************************************/

#ifndef _RPC_SMPL_SVR_H
#define _RPC_SMPL_SVR_H

typedef struct {
    UINT32  ulRPCServerID;
    UINT32  ulRPCServerVersion;
    UINT32  ServerStubTaskPriority;
    UINT32  ulStubStackSize;
    UINT32  ulMaxParamAreaSize;
    void    *pUserDefinedData;
} rpc_svr_config;

#ifdef __cplusplus
extern "C" {
#endif
UINT32 rpcsvr_SMPL_atof( rpc_server_stub_info * __pInfo );
UINT32 rpcsvr_SMPL_atoi( rpc_server_stub_info * __pInfo );
UINT32 rpcsvr_SMPL_ref_tsk2( rpc_server_stub_info * __pInfo );
```

```
UINT32 rpcsvr_SMPL_ref_sem2( rpc_server_stub_info * __pInfo );
INT32 rpcsvr_SMPL_init ( rpc_svr_config * __config );
INT32 rpcsvr_SMPL_shutdown ( UINT32 __ulServerID, UINT32
__ulServerVersion );
#ifdef __cplusplus
}
#endif

#endif /* End of _RPC_SMPL_SVR_H */
```

11.6 sample_svr.c

```
/*********************************************************************
  File Header Comment : Sample Config File for RPCGEN
*********************************************************************/

/*********************************************************************
 * Do not edit, unless you really know what you are doing,
 * this file was automatically generated by:
 *
 *    rpcgen.pl on 2007/11/28  12:54
 *
 * Be aware that this file may be generated as part of the build and
 * therefore any changes made by hand will possibly be lost on a
 * rebuild.
*********************************************************************/

#include <string.h>
#include "types.h"
#include "rpc_public.h"
#include <stdlib.h>
#include "kernel.h"
#include "sample_public.h"
#include "sample_svr.h"


#pragma section C_SAMPLE

UINT32 rpcsvr_SMPL_atof( rpc_server_stub_info * __pInfo )
{
     const char * nptr;
    UINT32 __ulInputParamOffset = 0UL;
    UINT32 __ulOutputParamCount = 0UL;
    double ret;
```

```
    nptr = ( const char *)( __pInfo->pucParamArea +
__ulInputParamOffset);


    ret = atof( nptr );


    *((double *)(__pInfo->pOutputIOVectorTable[ __ulOutputParamCount
].pBaseAddress)) = ret;
    __pInfo->pOutputIOVectorTable[ __ulOutputParamCount ].ulSize =
sizeof(double);


    __pInfo->ulOutputIOVectorTableSize = 1UL;


    return (UINT32)NULL;
}



UINT32 rpcsvr_SMPL_atoi( rpc_server_stub_info * __pInfo )
{
     const char * nptr;
    UINT32 __ulInputParamOffset = 0UL;
    int ret;


    nptr = *( const char **)( __pInfo->pucParamArea +
__ulInputParamOffset);


    ret = atoi( nptr );


    __pInfo->ulOutputIOVectorTableSize = 0UL;


    return ((UINT32)ret);
}



UINT32 rpcsvr_SMPL_ref_tsk2( rpc_server_stub_info * __pInfo )
{
    ID tskid;
```

```
     T_RTSK * pk_rtsk;
    UINT32 __ulInputParamOffset = 0UL;
    UINT32 __ulOutputParamCount = 0UL;
    ER ercd;

    tskid = *(ID *)( __pInfo->pucParamArea + __ulInputParamOffset);
    __ulInputParamOffset += ALIGNUP4(sizeof(ID));

    pk_rtsk = ( T_RTSK *)( __pInfo->pucParamArea +
__ulInputParamOffset);

    ercd = ref_tsk2( tskid, pk_rtsk );

    __pInfo->pOutputIOVectorTable[ __ulOutputParamCount
].pBaseAddress = pk_rtsk;
    __pInfo->pOutputIOVectorTable[ __ulOutputParamCount++ ].ulSize =
sizeof(*pk_rtsk);

    __pInfo->ulOutputIOVectorTableSize = 1UL;

    return ((UINT32)ercd);
}


UINT32 rpcsvr_SMPL_ref_sem2( rpc_server_stub_info * __pInfo )
{
    ID semid;
     T_RSEM * pk_rsem;
    UINT32 __ulInputParamOffset = 0UL;
    UINT32 __ulOutputParamCount = 0UL;
    ER ercd;

    semid = *(ID *)( __pInfo->pucParamArea + __ulInputParamOffset);
    __ulInputParamOffset += ALIGNUP4(sizeof(ID));
```

```
    pk_rsem = ( T_RSEM *)( __pInfo->pucParamArea +
__ulInputParamOffset);

    ercd = ref_sem2( semid, pk_rsem );

    __pInfo->pOutputIOVectorTable[ __ulOutputParamCount
].pBaseAddress = pk_rsem;
    __pInfo->pOutputIOVectorTable[ __ulOutputParamCount++ ].ulSize =
sizeof(*pk_rsem);

    __pInfo->ulOutputIOVectorTableSize = 1UL;

    return ((UINT32)ercd);
}


static UINT32 (* const __rpcsvr_SMPL_StubTable[])(rpc_server_stub_info
*) =
{
    rpcsvr_SMPL_atof,
    rpcsvr_SMPL_atoi,
    rpcsvr_SMPL_ref_tsk2,
    rpcsvr_SMPL_ref_sem2,
};


INT32 rpcsvr_SMPL_init ( rpc_svr_config * __config )
{
    rpc_server_info __server_info;

    if ( __config == NULL)
    {
        return -1L;
    }

    __server_info.ulRPCServerID             = RPCSVR_ID_SMPL;
```

```
    __server_info.ulRPCServerVersion        = RPCSVR_VERS_SMPL;
    __server_info.ServerStubTaskPriority    = __config-
>ServerStubTaskPriority;
    __server_info.ServerStubList            = __rpcsvr_SMPL_StubTable;
    __server_info.ulNumFunctions            = 4UL;
    __server_info.ulStubStackSize           = __config-
>ulStubStackSize;
    __server_info.pUserDefinedData          = __config-
>pUserDefinedData;
    __server_info.ulMaxParamAreaSize        = __config-
>ulMaxParamAreaSize;


    return rpc_start_server( &__server_info );
}


INT32 rpcsvr_SMPL_shutdown ( UINT32 __ulServerID, UINT32
__ulServerVersion )
{
    return rpc_stop_server( RPCSVR_ID_SMPL, RPCSVR_VERS_SMPL, NULL, 0UL
);
}
```

11.7 sample_public.h

```
/***********************************************************************
  File Header Comment : Sample Config File for RPCGEN
***********************************************************************/

/***********************************************************************
 * Do not edit, unless you really know what you are doing,
 * this file was automatically generated by:
 *
 *    rpcgen.pl on 2007/11/28  12:54
 *
 * Be aware that this file may be generated as part of the build and
 * therefore any changes made by hand will possibly be lost on a
 * rebuild.
***********************************************************************/

#ifndef _RPC_SMPL_PUBLIC_H
#define _RPC_SMPL_PUBLIC_H

#define RPCSVR_ID_SMPL      1
#define RPCSVR_VERS_SMPL        2

#endif /* End of _RPC_SMPL_PUBLIC_H */
```

Renesas Microcomputer Development Environment System
User's Manual
Stub Generator V.1.00

Publication Date: Rev.1.01, March 17, 2008
Published by: Sales Strategic Planning Div.
Renesas Technology Corp.
Edited by: Customer Support Department
Global Strategic Communication Div.
Renesas Solutions Corp.

Renesas Technology Corp. Sales Strategic Planning Div. Nippon Bldg., 2-6-2, Ohte-machi, Chiyoda-ku, Tokyo 100-0004, Japan

RENESAS

RENESAS SALES OFFICES

http://www.renesas.com

Refer to "**http://www.renesas.com/en/network**" for the latest and detailed information.

Renesas Technology America, Inc.
450 Holger Way, San Jose, CA 95134-1368, U.S.A
Tel: <1> (408) 382-7500, Fax: <1> (408) 382-7501

Renesas Technology Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K.
Tel: <44> (1628) 585-100, Fax: <44> (1628) 585-900

Renesas Technology (Shanghai) Co., Ltd.
Unit 204, 205, AZIACenter, No.1233 Lujiazui Ring Rd, Pudong District, Shanghai, China 200120
Tel: <86> (21) 5877-1818, Fax: <86> (21) 6887-7858/7898

Renesas Technology Hong Kong Ltd.
7th Floor, North Tower, World Finance Centre, Harbour City, Canton Road, Tsimshatsui, Kowloon, Hong Kong
Tel: <852> 2265-6688, Fax: <852> 2377-3473

Renesas Technology Taiwan Co., Ltd.
10th Floor, No.99, Fushing North Road, Taipei, Taiwan
Tel: <886> (2) 2715-2888, Fax: <886> (2) 3518-3399

Renesas Technology Singapore Pte. Ltd.
1 Harbour Front Avenue, #06-10, Keppel Bay Tower, Singapore 098632
Tel: <65> 6213-0200, Fax: <65> 6278-8001

Renesas Technology Korea Co., Ltd.
Kukje Center Bldg. 18th Fl., 191, 2-ka, Hangang-ro, Yongsan-ku, Seoul 140-702, Korea
Tel: <82> (2) 796-3115, Fax: <82> (2) 796-2145

Renesas Technology Malaysia Sdn. Bhd
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No.18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: <603> 7955-9390, Fax: <603> 7955-9510

Stub Generator V.1.00 User's Manual

Renesas Technology Corp.
2-6-2, Ote-machi, Chiyoda-ku, Tokyo, 100-0004, Japan