お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

User's Manual

スタブジェネレータ V.1.00

ユーザーズマニュアル

ルネサスマイクロコンピュータ開発環境システム

ルネサス エレクトロニクス
www.renesas.com

Rev.1.01　2008.03

本資料ご利用に際しての留意事項

１．本資料は、お客様に用途に応じた適切な弊社製品をご購入いただくための参考資料であり、本資料中に記載の技術情報について弊社または第三者の知的財産権その他の権利の実施、使用を許諾または保証するものではありません。
２．本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例など全ての情報の使用に起因する損害、第三者の知的財産権その他の権利に対する侵害に関し、弊社は責任を負いません。
３．本資料に記載の製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的、あるいはその他軍事用途の目的で使用しないでください。また、輸出に際しては、「外国為替および外国貿易法」その他輸出関連法令を遵守し、それらの定めるところにより必要な手続を行ってください。
４．本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例などの全ての情報は本資料発行時点のものであり、弊社は本資料に記載した製品または仕様等を予告なしに変更することがあります。弊社の半導体製品のご購入およびご使用に当たりましては、事前に弊社営業窓口で最新の情報をご確認いただきますとともに、弊社ホームページ(http://www.renesas.com)などを通じて公開される情報に常にご注意ください。
５．本資料に記載した情報は、正確を期すため慎重に制作したものですが、万一本資料の記述の誤りに起因する損害がお客様に生じた場合においても、弊社はその責任を負いません。
６．本資料に記載の製品データ、図、表などに示す技術的な内容、プログラム、アルゴリズムその他応用回路例などの情報を流用する場合は、流用する情報を単独で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。弊社は、適用可否に対する責任は負いません。
７．本資料に記載された製品は、各種安全装置や運輸・交通用、医療用、燃焼制御用、航空宇宙用、原子力、海底中継用の機器・システムなど、その故障や誤動作が直接人命を脅かしあるいは人体に危害を及ぼすおそれのあるような機器・システムや特に高度な品質・信頼性が要求される機器・システムでの使用を意図して設計、製造されたものではありません（弊社が自動車用と指定する製品を自動車に使用する場合を除きます）。これらの用途に利用されることをご検討の際には、必ず事前に弊社営業窓口へご照会ください。なお、上記用途に使用されたことにより発生した損害等について弊社はその責任を負いかねますのでご了承願います。
８．第７項にかかわらず、本資料に記載された製品は、下記の用途には使用しないでください。これらの用途に使用されたことにより発生した損害等につきましては、弊社は一切の責任を負いません。
　１）生命維持装置。
　２）人体に埋め込み使用するもの。
　３）治療行為（患部切り出し、薬剤投与等）を行うもの。
　４）その他、直接人命に影響を与えるもの。
９．本資料に記載された製品のご使用につき、特に最大定格、動作電源電圧範囲、放熱特性、実装条件およびその他諸条件につきましては、弊社保証範囲内でご使用ください。弊社保証値を越えて製品をご使用された場合の故障および事故につきましては、弊社はその責任を負いません。
１０．弊社は製品の品質および信頼性の向上に努めておりますが、特に半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。弊社製品の故障または誤動作が生じた場合も人身事故、火災事故、社会的損害などを生じさせないよう、お客様の責任において冗長設計、延焼対策設計、誤動作防止設計などの安全設計（含むハードウェアおよびソフトウェア）およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特にマイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
１１．本資料に記載の製品は、これを搭載した製品から剥がれた場合、幼児が口に入れて誤飲する等の事故の危険性があります。お客様の製品への実装後に容易に本製品が剥がれることがなきよう、お客様の責任において十分な安全設計をお願いします。お客様の製品から剥がれた場合の事故につきましては、弊社はその責任を負いません。
１２．本資料の全部または一部を弊社の文書による事前の承諾なしに転載または複製することを固くお断りいたします。
１３．本資料に関する詳細についてのお問い合わせ、その他お気付きの点等がございましたら弊社営業窓口までご照会ください。

はじめに

本マニュアルは、「スタブジェネレータ」の使用方法を述べたものです。ご使用になる前に本マニュアルを良く読んで理解してください。

■表記上の注意事項

- RPCGEN
  本書では、スタブジェネレータをRPCGENと略記します。
- 数値のプリフィックス
  "0x"は16進数を意味します。プリフィックスの無い場合は10進数です。
- ディレクトリ区切り記号は”¥”です。
- [メニュー->メニューオプション]
  “->”はメニューオプションを示します。(例[ファイル->保存])
- $(xxxx)
  $(xxxx)は、High-performance Embedded Workshopのカスタムプレースホルダを示します。

■商標等

すべての商標および登録商標は、それぞれの所有者に帰属します。

- TRON は、"The Real-time Operating system Nucleus" の略称です。ITRON は、"Industrial TRON" の略称です。μITRON は、"Micro Industrial TRON" の略称です。TRON、ITRON、および μITRON は、コンピュータの仕様に対する名称であり、特定の商品ないし商品群を指すものではありません。μITRON4.0 仕様は、(社)トロン協会が策定したオープンなリアルタイムカーネル仕様です。μITRON4.0 仕様の仕様書は、(社)トロン協会ホームページ(http://www.assoc.tron.org/)から入手が可能です。μITRON 仕様の著作権は(社)トロン協会に属しています。
- Microsoft、Windows は米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。Windows の正式名称は、Microsoft Windows Operating System です。
- その他、本書で登場するシステム名、製品名は各社の登録商標または商標です。

■ホームページ

弊社ホームページにて各種サポート情報をお知らせしておりますので、あわせてご利用ください。
http://japan.renesas.com/homepage.jsp

目次

1. 概要

1.1 概要

スタブジェネレータ(以下、RPCGEN と略します)は、弊社製リアルタイム OS の RPC(Remote Procedure Call)機能を使用する際に必要となるクライアント・サーバスタブのソースコードを生成するツールです。テキストエディタを用いて Config ファイルを作成し、これを RPCGEN に入力することで、クライアント・サーバスタブのソースコードが生成されます。

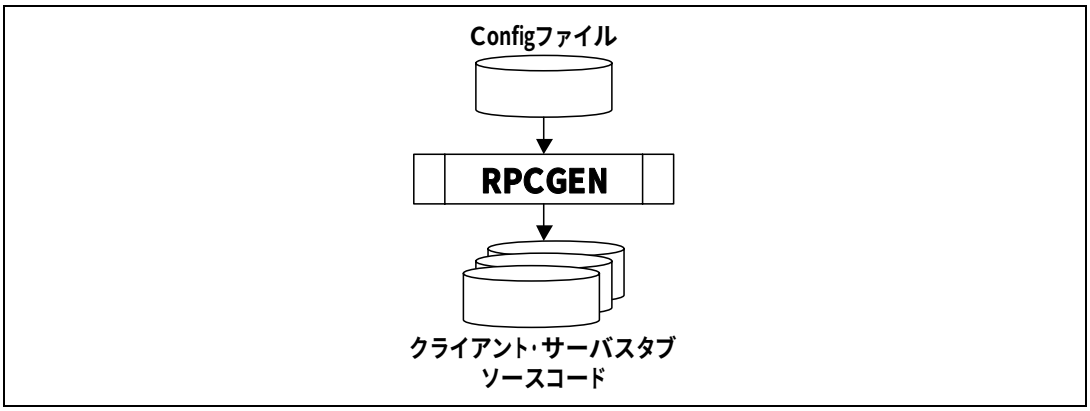

図 1.1 RPCGEN 概要

1.2 動作環境

RPCGEN は、Perl スクリプトファイルとして提供されます。
表 1.1に、動作環境を示します。

表1.1 動作環境

項目	動作環境
対象リアルタイム OS	HI7200/MP V.1.00 Release 00 以降
Perl 実行環境	弊社が動作確認に使用したものは、ActiveState 社の ActivePerl 5.8.8.820(Windows®(x86)版)です。 ActivePerl は、以下の Web サイトから無償ダウンロード可能です。 http://www.activestate.com/Products/activeperl/
ホスト PC	以下の OS を搭載したもの ・Windows XP (32bit 版) ・Windows 2000

2. インストール

2.1 入手

RPCGEN は、弊社 Web サイトから無償でダウンロードできます。以下の URL からダウンロードしてください。

http://japan.renesas.com/rpcgen/

2.2 インストール

ダウンロードファイルは、zip 形式で圧縮されています。これを解凍すると、表 2.1に示すファイルが復元されます。これらを適切なフォルダに格納してください。

表2.1 RPCGEN 提供ファイル

項番	ファイル名	内容
1	rpcgen.pl	RPCGEN 本体
2	rpcgen_clnt.pm	RPCGEN 本体が使用するパッケージモジュール
3	rpcgen_svr.pm	
4	rpcgen_info.pm	
5	rpcgen_misc.pm	

3. RPCGEN が生成する関数とファイル

3.1 RPCGEN が生成する関数

図 3.1に、RPCGEN が生成する関数を示します。

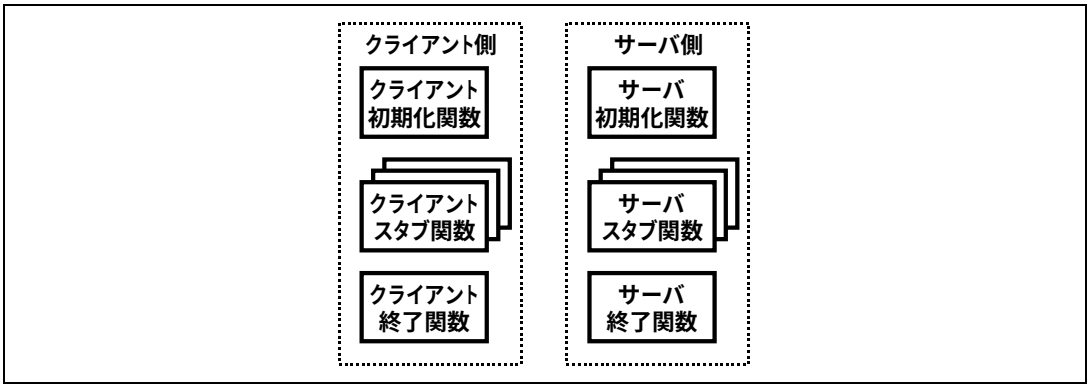

図 3.1 RPCGEN が生成する関数

3.1.1 クライアント側

(1) クライアント初期化関数

サーバと接続する関数です。CLNT_NOINIT 指定により、クライアント初期化関数を生成しないようにすることもできます。

(2) クライアント終了関数

サーバとの接続を切断する関数です。CLNT_NOSHUTDOWN 指定により、クライアント終了関数を生成しないようにすることもできます。

(3) クライアントスタブ関数

サーバ関数と同じまたは同様の API を持ち、RPC コールを行う関数です。SVRAUTH 指定がない場合は、サーバ関数と同じ API となります。CLNTSTUB 指定により、特定のクライアントスタブ関数だけを生成しないようにすることもできます。

3.1.2 サーバ側

(1) サーバ初期化関数

サーバを開始する関数です。SVR_NOINIT 指定により、サーバ初期化関数を生成しないようにすることもできます。

(2) サーバ終了関数

サーバを停止する関数です。SVR_NOSHUTDOWN 指定により、サーバ終了関数を生成しないようにすることもできます。

(3) サーバスタブ関数

RPC ライブラリからコールバックされ、サーバ関数を呼び出す関数です。SVRSTUB 指定により、特定のサーバスタブ関数だけを生成しないようにすることもできます。また、SVRFUNC 指定により、呼び出すサーバ関数を別名に置き換えることができます。

3.2 RPCGEN が生成するファイル

RPCGEN は、表 3.1に示すファイルを Config ファイル名に従って生成します。

表3.1 RPCGEN 生成ファイル

項番	生成ファイル名	内容	生成場所の指定方法
1	<Config ファイル>_clnt.c	クライアントスタブソースファイル	CLNTS オプションまたは Config ファイルでの CLNT_SOURCEPATH 定義
2	<Config ファイル>_clnt.h	クライアントスタブヘッダファイル	CLNTI オプションまたは Config ファイルでの CLNT_INCPATH 定義
3	<Config ファイル>_private.h	クライアントスタブ用内部ヘッダファイル	
4	<Config ファイル>_svr.c	サーバスタブソースファイル	SVRS オプションまたは Config ファイルでの SVR_SOURCEPATH 定義
5	<Config ファイル>_svr.h	サーバスタブヘッダファイル	SVRI オプションまたは Config ファイルでの SVR_INCPATH 定義
6	<Config ファイル>_public.h	クライアント・サーバスタブ用ヘッダファイル	PUBI オプションまたは Config ファイルでの PUB_INCPATH 定義

例えば、Config ファイル名が”sample.x”の場合、以下のファイルを生成します。
sample_clnt.c、sample_clnt.h、sample_svr.c、sample_svr.h、sample_private.h、sample_public.h

また、RPCGEN は生成途中の情報を一時的に退避するため、上記表の生成ファイル名の先頭に”__”(アンダースコア 2 つ)を付加した中間ファイルをカレントディレクトリに生成します。

(1) <Config ファイル>_clnt.c(クライアントスタブソースファイル)

クライアントスタブ関数、クライアント初期化関数、クライアント終了関数が出力されます。

(2) <Config ファイル>_clnt.h(クライアントスタブヘッダファイル)

アプリケーションからクライアントスタブ関数、クライアント初期化関数、クライアント終了関数を利用するための定義が出力されます。

クライアントスタブ関数、クライアント初期化関数、クライアント終了関数のプロトタイプ宣言などが含まれます。

(3) <Config ファイル>_private.h(クライアントスタブ用内部ヘッダファイル)

<Config ファイル>_clnt.c のみがインクルードするヘッダファイルです。

(4) <Config ファイル>_svr.c(サーバスタブソースファイル)

サーバスタブ関数、サーバ初期化関数、サーバ終了関数、サーバスタブ関数テーブルが出力されます。

(5) <Config ファイル>_svr.h(サーバスタブヘッダファイル)

アプリケーションからサーバ初期化関数、サーバ終了関数を利用するための定義が出力されます。

サーバスタブ関数、サーバ初期化関数、サーバ終了関数のプロトタイプ宣言などが含まれます。

(6) <Config ファイル>_public.h(クライアント・サーバスタブ用ヘッダファイル)

クライアント側およびサーバ側のアプリケーション用のヘッダファイルです。

「8.9 <Config ファイル>_public.h」を参照してください。

4. RPCGEN の実行

4.1 RPCGEN の実行

RPCGEN は、コマンドプロンプトで実行します。

```
perl△-I<rpcgen パス>△<rpcgen パス>¥rpcgen.pl△<Config ファイル>[△<オプション>...](RET)
```

<rpcgen パス>は、RPCGEN をインストールしたパスです。

<Config ファイル>は、クライアント・サーバスタブの生成仕様を記述したファイルで、ユーザが作成します。Config ファイルの拡張子は、標準的には".x"を使用しますが、コマンドラインで指定するときには拡張子を省略することはできません。

上記コマンドの実行により、RPCGEN は<Config ファイル>に記載された内容に従ってクライアント・サーバスタブの C 言語ソースファイルおよびヘッダファイルを生成します。

また、<Config ファイル>, <オプション>を省略すると、コマンドシンタックスが表示されます。

4.2 オプション解説

4.2.1 PUBI オプション

■書式

```
PUBI="<パス>"
```

■解説

<Config ファイル>_public.h を出力するパスを指定します。
本オプション指定時は、Config ファイルに記述した PUB_INCPATH 定義は無視されます。

4.2.2 CLNTS オプション

■書式

```
CLNTS="<パス>"
```

■解説

<Config ファイル>_clnt.c を出力するパスを指定します。
本オプション指定時は、Config ファイルに記述した CLNT_SOURCEPATH 定義は無視されます。

4.2.3 CLNTI オプション

■書式

```
CLNTI="<パス>"
```

■解説

<Config ファイル>_clnt.h および<Config ファイル>_private.h を出力するパスを指定します。
本オプション指定時は、Config ファイルに記述した CLNT_INCPATH 定義は無視されます。

4.2.4 SVRS オプション

■書式

```
SVRS="<パス>"
```

■解説

<Config ファイル>_svr.c を出力するパスを指定します。
本オプション指定時は、Config ファイルに記述した SVR_SOURCEPATH 定義は無視されます。

4.2.5 SVRI オプション

■書式

```
SVRI="<パス>"
```

■解説

<Config ファイル>_svr.h を出力するパスを指定します。
本オプション指定時は、Config ファイルに記述した SVR_INCPATH 定義は無視されます。

4.3 High-performance Embedded Workshop のビルドフェーズでの実行

ワークスペースに RPCGEN をカスタムビルドフェーズとして登録すると、ビルド時に RPCGEN を自動的に実行できるようになります。

(1) カスタムプレースホルダの設定

カスタムプレースホルダとして、以下の 2 つを定義します。

(a) RPCGEN のパス
[プレースホルダ]：RPCGEN_INST
[説明]：RPCGEN Base directory
[ディレクトリ]：RPCGEN が格納されているディレクトリ

(b) Perl.exe のパス
[プレースホルダ]：PERL_INST
[説明]：Perl Base directory
[ディレクトリ]：perl.exe が格納されているディレクトリ

カスタムプレースホルダを定義するには、次のように操作します。
High-performance Embedded Workshop のメニューバー[基本設定→カスタマイズ]より[プレースホルダ]タブを選びます。そして、[アプリケーション内有効カスタムプレースホルダ]または[ワークスペース内有効カスタムプレースホルダ]の[追加]ボタンを押してください。これにより、図 4.1に示す[新規カスタムプレースボルダ]ダイアログボックスが開きますので、ここで上記の内容を設定してください。

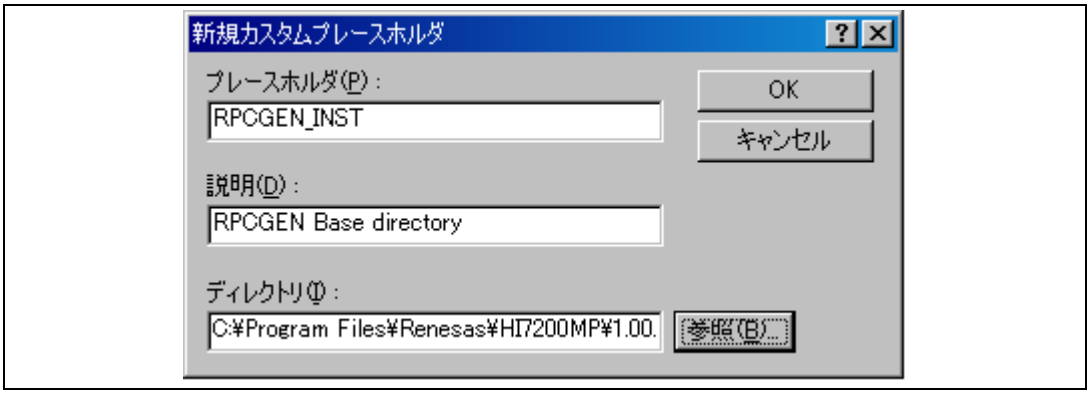

図 4.1 [新規カスタムプレースホルダ]ダイアログボックス

(2) ファイル拡張子の登録

カスタムビルドフェーズを機能させるには、カスタムビルドフェーズで扱うファイルの拡張子として、Config ファイルの拡張子".x"を登録する必要があります。

High-performance Embedded Workshop メニューバーより[プロジェクト→ファイルの拡張子]を選択すると、図 4.2の[ファイル拡張子]ダイアログボックスが開きます。

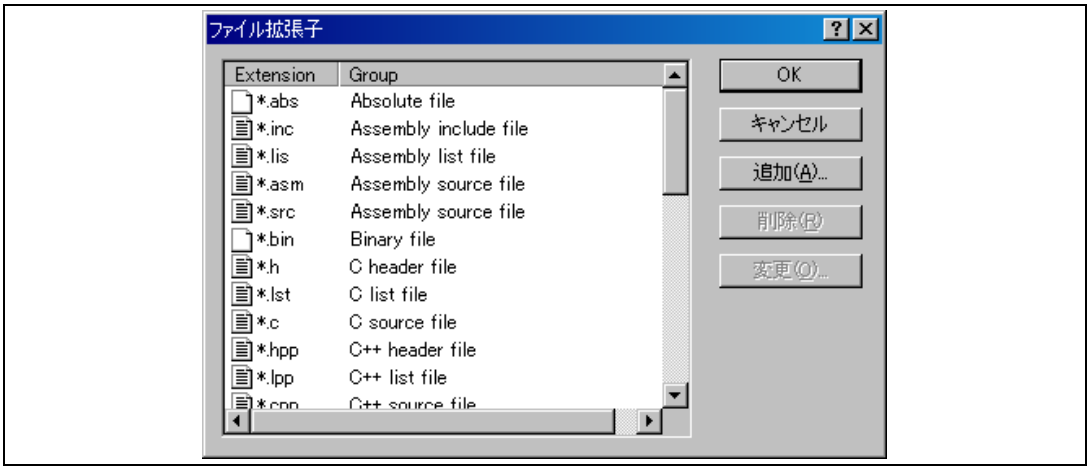

図 4.2 [ファイル拡張子]ダイアログボックス

ここで、[追加]ボタンを押すと、図 4.3の[ファイル拡張子の追加]ダイアログボックスが開きますので、図 4.3に示すように拡張子".x"を登録してください。

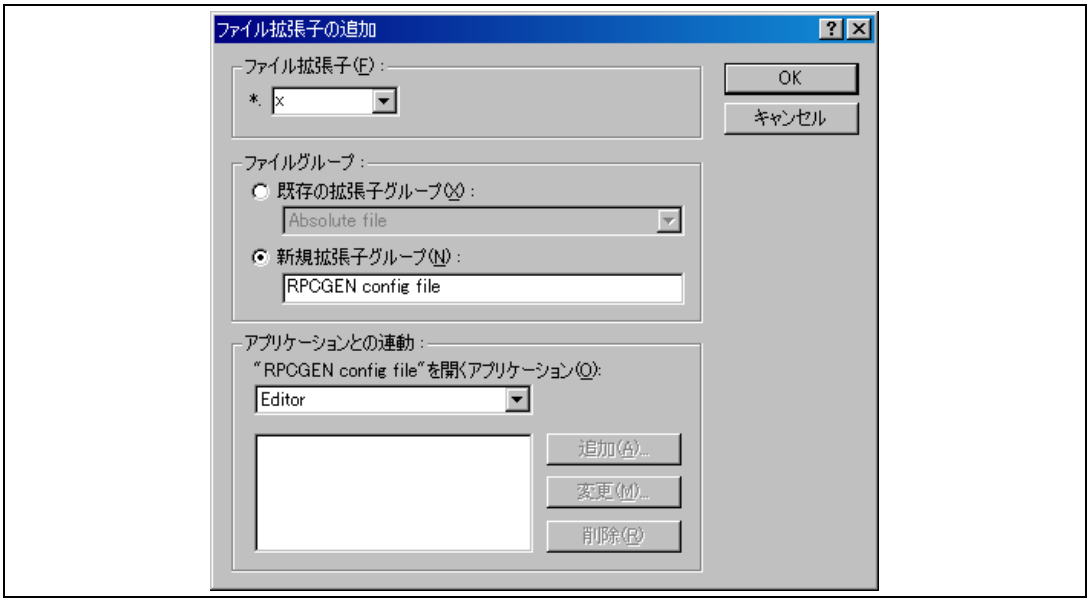

図 4.3 [ファイル拡張子の追加]ダイアログボックス

(3) RPCGEN カスタムビルドフェーズの作成

High-performance Embedded Workshop メニューバーより[ビルド→ビルドフェーズ]を選択すると、図 4.4の[ビルドフェーズ]ダイアログボックスが開きます。

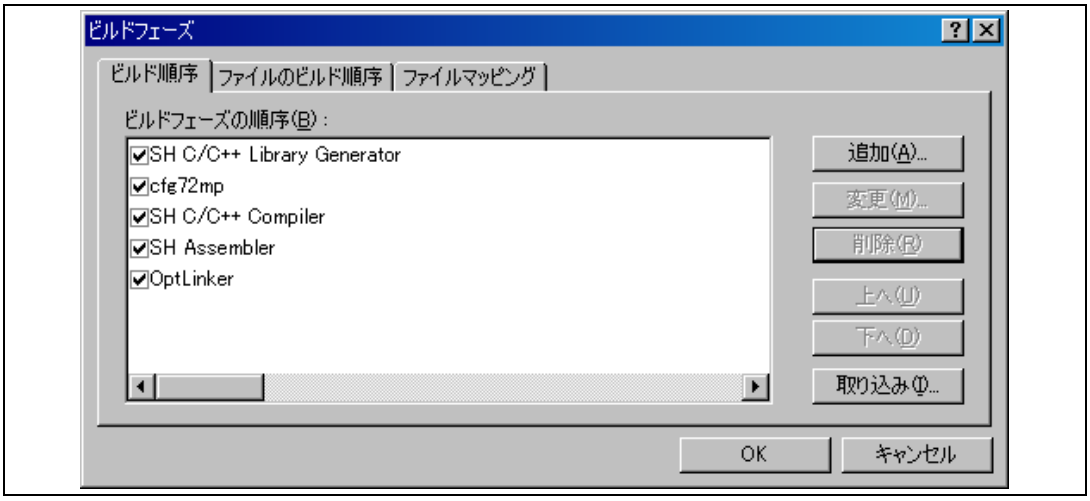

図 4.4 [ビルドフェーズ]ダイアログボックス

ここで[追加]ボタンを押すと、新規ビルドフェーズを登録するウィザードが始まります(図 4.5)。

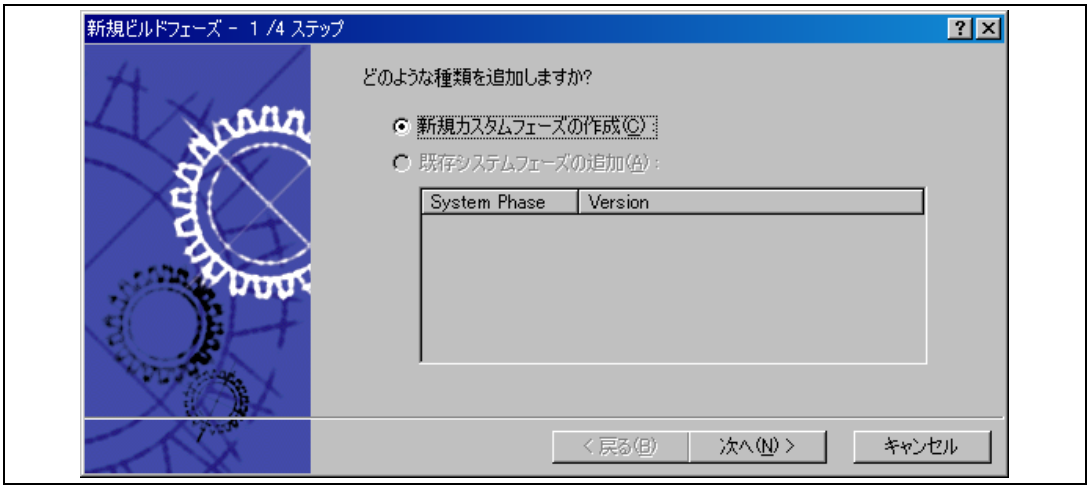

図 4.5 [新規ビルドフェーズ - 1/4 ステップ]ダイアログボックス

[次へ]を押すと 2/4 ステップに進みます。

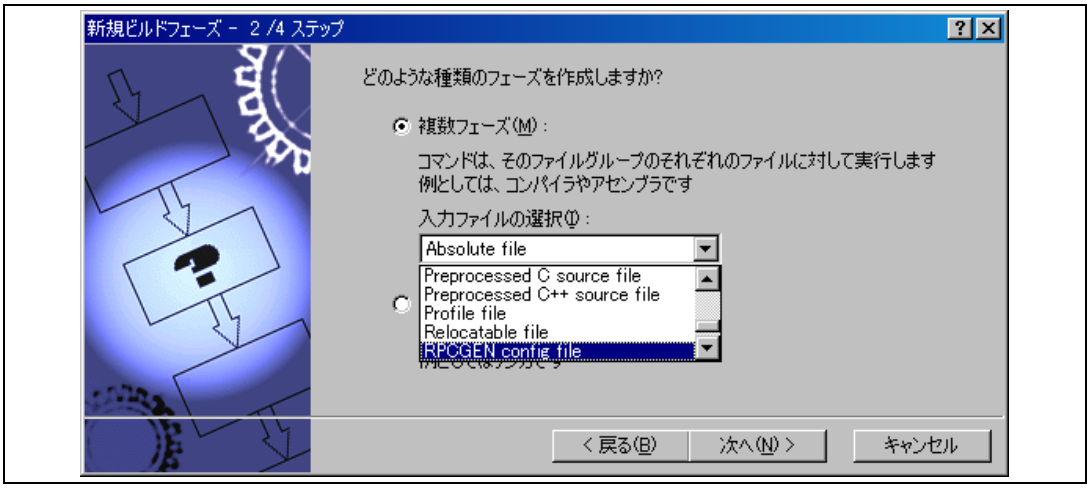

図 4.6 [新規ビルドフェーズ - 2/4 ステップ]ダイアログボックス

[2/4 ステップ]では、[複数フェーズ]を選択し、[入力ファイルの選択]で”RPCGEN config file”を選択してください。その後、[次へ]押すと 3/4 ステップに進みます。

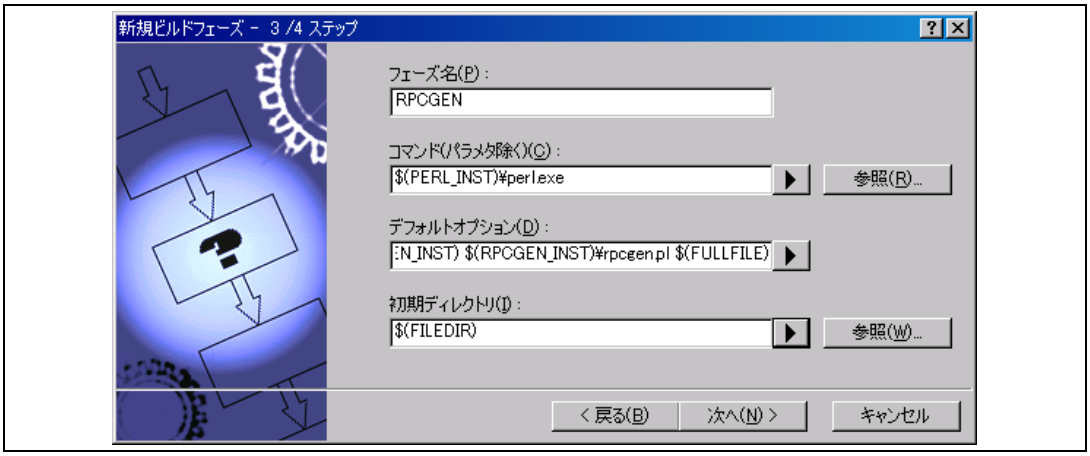

図 4.7 [新規ビルドフェーズ - 3/4 ステップ]ダイアログボックス

[フェーズ名]設定はユーザ任意ですが、本例では”rpcgen”としています。
[コマンド]設定は、”$(PERL_INST)¥perl.exe”を指定してください。
[デフォルトオプション]設定は、"–I $(RPCGEN_INST) $(RPCGEN_INST)¥rpcgen.pl $(FULLFILE)"を指定してください。
[初期ディレクトリ]設定は、$(FILEDIR)を指定してください。
[次へ]を押すと、4/4 ステップに進みます。

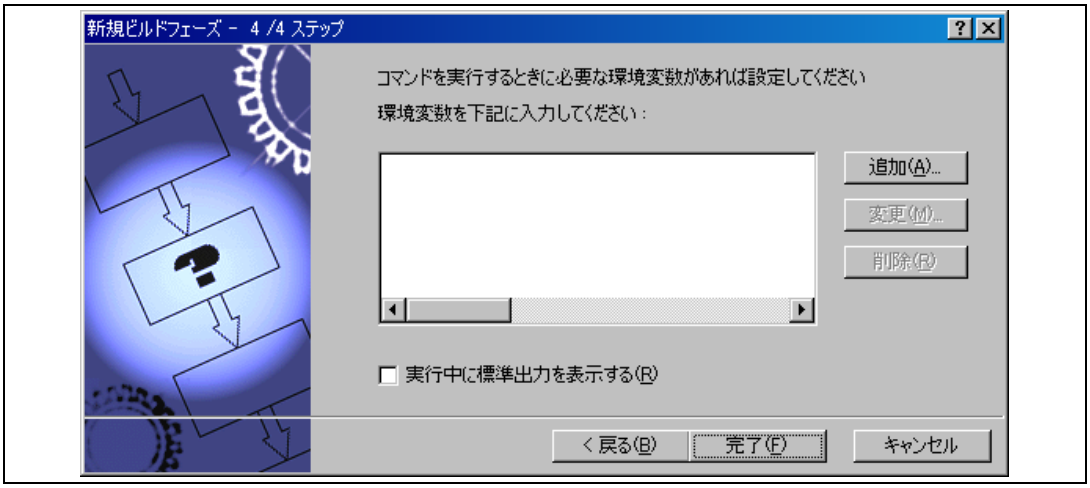

図 4.8 [新規ビルドフェーズ - 4/4 ステップ]ダイアログボックス

RPCGEN としての環境変数の設定は、特に不要です。必要に応じて、Perl 実行環境としての環境変数を設定してください。

以上で、RPCGEN カスタムビルドフェーズの作成は完了です。

次に RPCGEN の出力メッセージシンタックスを設定します。これにより、High-performance Embedded Workshop の[Build]ウィンドウに出力される RPCGEN のエラーメッセージをダブルクリックすることで、Config ファイル(*.x)を表示できるようになります。
図 4.4の[ビルドフェーズ]ダイアログボックスで、"rpcgen"を選択し[変更]ボタンを押してください。[rpcgen の変更]ダイアログボックスが開きます。
ここで[出力シンタックス]タブを選び、図 4.9のようにエラーのシンタックスを設定してください。

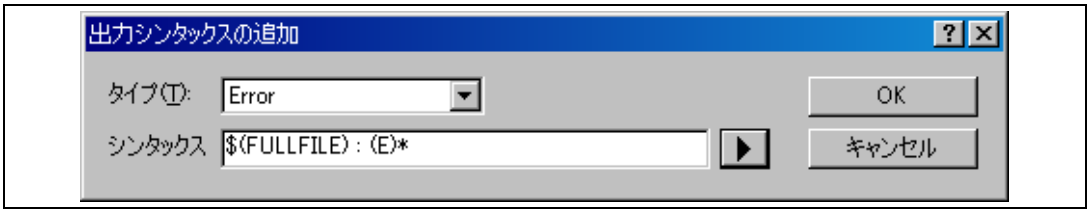

図 4.9 [出力シンタックスの追加]ダイアログボックス

次に、High-performance Embedded Workshop の「クリーンアクティブプロジェクト」および「クリーン全プロジェクト」によって、RPCGEN 出力ファイルも削除されるように設定します。

[ビルド→rpcgen]メニューを選択し、図 4.10の[rpcgen Options]ダイアログボックスを開いてください。[出力ファイル]タブを選び、”RPCGEN config file”のフォルダアイコンを選択している状態で、[追加]ボタンを押し、図 4.11の[出力ファイルの追加]ダイアログボックスで RPCGEN が生成するファイルを登録します。「3.2 RPCGEN が生成するファイル」を参考に設定してください。

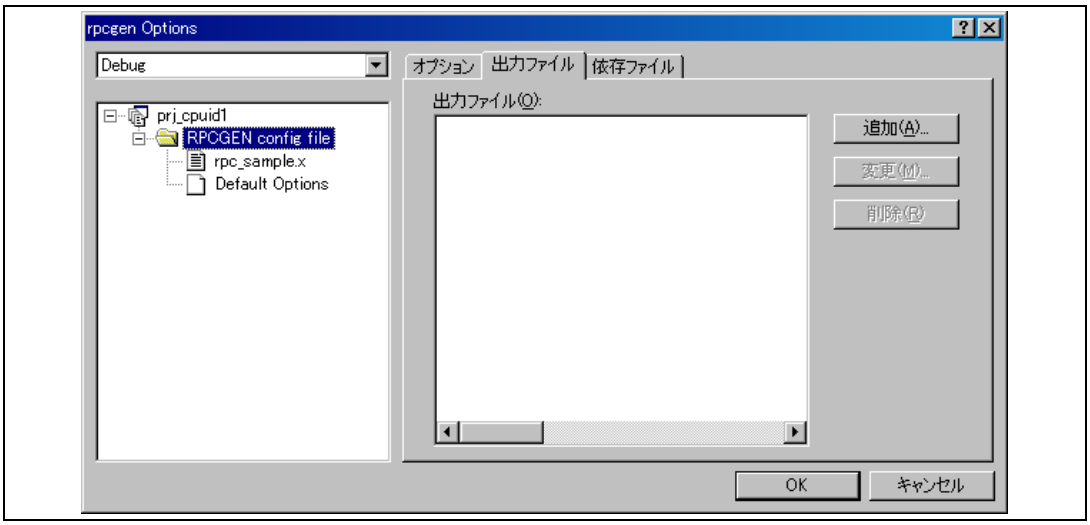

図 4.10 [rpcgen Options]ダイアログボックス

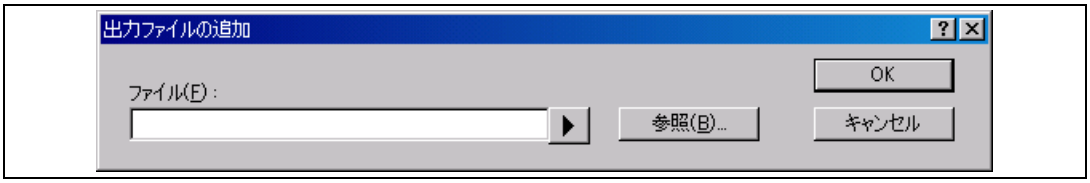

図 4.11 [出力ファイルの追加]ダイアログボックス

(4) ビルドフェーズの設定

作成した RPCGEN カスタムビルドフェーズは最下位に表示されます。これを以下のように”rpcgen”が”SH C/C++ Library Generator”より上位になるよう、[上へ]ボタンを使って移動させてください。

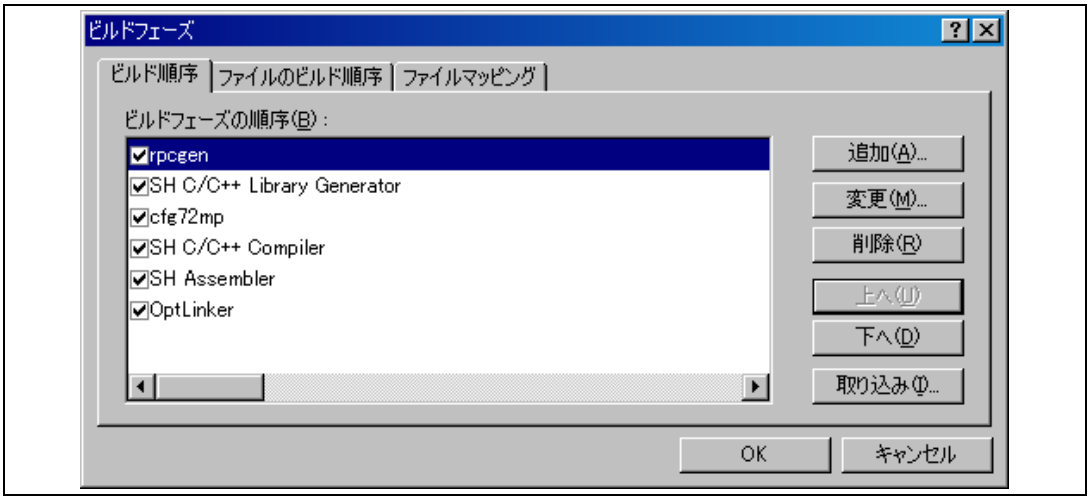

図 4.12 [ビルドフェーズ]ダイアログボックスの[ビルド順序]タブ

次に、[ファイルのビルド順序]タブにおいて、[ファイルグループ]から”RPCGEN config file”を選択し、右側の[フェーズの順序]の[rpcgen]をチェックしてください。

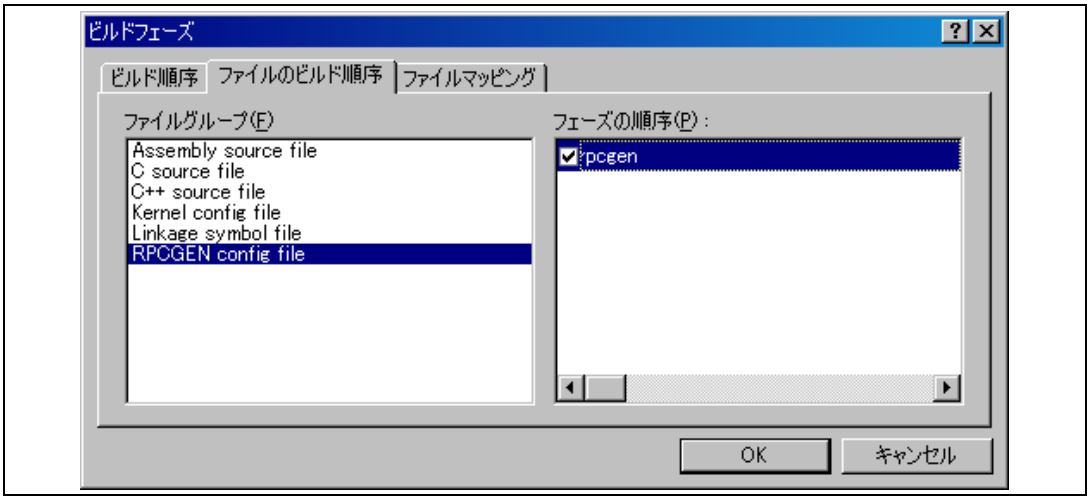

図 4.13 [ビルドフェーズ]ダイアログボックスの[ファイルのビルド順序]タブ

以上で RPCGEN のビルドフェーズ設定は完了です。

5. Config ファイル記述仕様

Config ファイルには、クライアント・サーバスタブを生成するための各種情報を定義します。具体的には、サーバ関数の API 情報に RPCGEN に渡すキーワードを付加する形式で行います。以降の説明を理解し、テキスト形式の Config ファイルを作成してください。Config ファイルの拡張子は".x"です。

ひとつの Config ファイルでひとつのサーバを定義できます。複数のサーバを同じ Config ファイルに定義することはできません。

5.1 定義文

定義文は、以下のいずれかの書式を基本とします。

```
<キーワード>{・・・};
<キーワード>;
```

中カッコで括られた中には複数の定義が可能なものも存在します。以下に例を挙げます。

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

定義文には、以下のものがあります。

(1) RPCGEN が生成するファイルの格納場所の定義

- PUB_INCPATH{ … }; <Config ファイル>_public.h の生成パスの定義
- CLNT_SOURCEPATH{ … }; <Config ファイル>_clnt.c の生成パスの定義
- CLNT_INCPATH{ … }; <Config ファイル>_clnt.h および<Config ファイル>_private.h の生成パスの定義
- SVR_SOURCEPATH{ … }; <Config ファイル>_svr.c の生成パスの定義
- SVR_INCPATH{ … }; <Config ファイル>_svr.h の生成パスの定義

(2) RPCGEN が生成するファイルへの#include 文の出力制御

- GLOBAL_INCFILE{ … }; クライアント・サーバスタブ共通のインクルードファイルの定義
- CLNT_INCFILE{ … }; クライアントスタブ用インクルードファイルの定義
- SVR_INCFILE{ … }; サーバスタブ用インクルードファイルの定義

(3) サーバ情報の定義

- SVR_NAME{ … }; サーバ名称の定義
- SVR_ID{ … }; サーバ ID の定義
- SVR_VERSION{ … }; サーババージョンの定義
- SVR_NOINIT; サーバ初期化関数を生成しない
- SVR_NOSHUTDOWN; サーバ終了関数を生成しない
- SVR_NOSTUBTBL; サーバスタブ関数テーブルを生成しない
- SVR_STATIC{ … }; スタティックサーバの使用
- SVR_SECTION{ … }; サーバスタブに付与するセクション名の定義
- SVR_AUTH; サーバ ID・サーババージョンの与え方

(4) クライアント情報の定義

- CLNT_NOINIT; クライアント初期化関数を生成しない
- CLNT_NOSHUTDOWN; クライアント終了関数を生成しない
- CLNT_CALLCHK; RPC コールのリターン値の保存
- CLNT_SECTION{ … }; クライアントスタブに付与するセクション名の定義

(5) サーバ関数の定義

- RPC_FUNC{ … }; サーバ関数の定義

RPC_FUNC のみ、{ }内に複数のサーバ関数の定義文を含みます。

5.2 コメント

Config ファイルでのコメント記法は、通常の C/C++言語コメントと同じです。

Config ファイルのコメントとして扱われた部分は、最初に有効なキーワードが定義されるまでのコメントに限り、RPCGEN が生成する全てのソース・ヘッダファイルの先頭に出力されます。

5.3 ファイルの格納場所の定義文

以下の定義文によって、RPCGEN が生成するファイルのパスを定義します。以下の定義文を省略した場合は、RPCGEN はカレントディレクトリに該当するソース・ヘッダファイルを生成します。

なお、定義が存在する場合でも、そのパスが存在しない場合、またはそのパスに書換えが禁止された出力ファイルが既に存在する場合、エラーメッセージを表示し処理を中断します。

このカテゴリには、以下の定義文があります。

- PUB_INCPATH{ … }; <Config ファイル>_public.h の生成パスの定義
- CLNT_SOURCEPATH{ … }; <Config ファイル>_clnt.c の生成パスの定義
- CLNT_INCPATH{ … }; <Config ファイル>_clnt.h および<Config ファイル>_private.h の生成パスの定義
- SVR_SOURCEPATH{ … }; <Config ファイル>_svr.c の生成パスの定義
- SVR_INCPATH{ … }; <Config ファイル>_svr.h の生成パスの定義

5.3.1 PUB_INCPATH

■書式

```
PUB_INCPATH{"<パス>"};
```

■解説

<Config ファイル>_public.h を生成するパスを定義します。

PUBI オプション指定時は、本定義文は無視されます。

■使用例

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

カレントディレクトリの下の public_include ディレクトリに<Config ファイル>_public.h を生成します。

5.3.2 CLNT_SOURCEPATH

■書式

```
CLNT_SOURCEPATH{"<パス>"};
```

■解説

<Config ファイル>_clnt.c を生成するパスを定義します。

CLNTS オプション指定時は、本定義文は無視されます。

■使用例

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

カレントディレクトリの下の clnt¥source ディレクトリに<Config ファイル>_clnt.c を生成します。

5.3.3 CLNT_INCPATH

■書式

```
CLNT_INCPATH{"<パス>"};
```

■解説

<Config ファイル>_clnt.h および<Config ファイル>_private.h を生成するパスを定義します。
CLNTI オプション指定時は、本定義文は無視されます。

■使用例

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

C:¥clnt¥include ディレクトリに<Config ファイル>_clnt.h および<Config ファイル>_private.h を生成します。

5.3.4 SVR_SOURCEPATH

■書式

```
SVR_SOURCEPATH{"<パス>"};
```

■解説

<Config ファイル>_svr.c を生成するパスを定義します。
SVRS オプション指定時は、本定義文は無視されます。

■使用例

```
SVR_SOURCEPATH{"svr¥source"};
```

カレントディレクトリの下の svr¥source ディレクトリに<Config ファイル>_svr.c を生成します。

5.3.5 SVR_INCPATH

■書式

```
SVR_INCPATH{"<パス>"};
```

■解説

<Config ファイル>_svr.h を生成するパスを定義します。
SVRI オプション指定時は、本定義文は無視されます。

■使用例

```
SVR_INCPATH{"svr¥include"};
```

カレントディレクトリの下の svr¥include ディレクトリに<Config ファイル>_svr.h を生成します。

5.4 #include 文の出力制御の定義文

本節に記載の定義文によって、RPCGEN が生成するファイルに、ユーザ作成ファイルの#include 文を出力できます。

なお、表 5.1に示すファイルの#include 文は常に出力されます。

表5.1 デフォルトのインクルードファイル

項番	インクルードファイル	解説
1	string.h	コンパイラ標準ヘッダファイル
2	types.h	弊社製 OS(HI7200/MP など)に付属するデータタイプヘッダファイル
3	rpc_public.h	弊社製 OS(HI7200/MP など)に付属する RPC ヘッダファイル

このカテゴリには、以下の定義文があります。

- GLOBAL_INCFILE{ … }; クライアント・サーバスタブ共通のインクルードファイルの定義
- CLNT_INCFILE{ … }; クライアントスタブ用インクルードファイルの定義
- SVR_INCFILE{ … }; サーバスタブ用インクルードファイルの定義

5.4.1 インクルード順序

RPCGEN は以下の順序に従って#include 文を出力します。

(1) <Config ファイル>_clnt.c

1 デフォルトのインクルードファイル(表 5.1の順序)
2 GLOBAL_INCFILE で定義されたファイル(複数指定した場合は、その指定順)
3 CLNT_INCFILE で定義されたファイル(複数指定した場合は、その指定順)
4 RPCGEN が生成した<Config ファイル>_public.h(SVR_AUTH 定義がない場合のみ)
5 RPCGEN が生成した<Config ファイル>_clnt.h
6 RPCGEN が生成した<Config ファイル>_private.h

(2) <Config ファイル>_svr.c

1 デフォルトのインクルードファイル(表 5.1の順序)
2 GLOBAL_INCFILE で定義されたファイル(複数指定した場合は、その指定順)
3 SVR_INCFILE で定義されたファイル(複数指定した場合は、その指定順)
4 RPCGEN が生成した<Config ファイル>_ public.h(SVR_AUTH 定義がない場合のみ)
5 RPCGEN が生成した<Config ファイル>_svr.h

5.4.2 GLOBAL_INCFILE

■書式

```
GLOBAL_INCFILE{<ファイル指定>[△<ファイル指定>…]};
```

■解説

<Config ファイル>_clnt.c および<Config ファイル>_svr.c に、指定されたファイルの#include 文を出力します。<ファイル指定>は、ファイル名を" "または< >で括った形式で記述します。両者の相違は、#include 文の仕様の通りです。

複数ファイルを指定する場合は、スペースで区切ります。

■使用例

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

5.4.3 CLNT_INCFILE

■書式

```
CLNT_INCFILE{<ファイル指定>[△<ファイル指定>…]};
```

■解説

<Config ファイル>_clnt.c に、指定されたファイルの#include 文を出力します。<ファイル指定>は、ファイル名を" "または< >で括った形式で記述します。両者の相違は、#include 文の仕様の通りです。

複数ファイルを指定する場合は、スペースで区切ります。

■使用例

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

5.4.4 SVR_INCFILE

■書式

```
SVR_INCFILE{<ファイル指定>[△<ファイル指定>…]};
```

■解説

<Config ファイル>_svr.c に、指定されたファイルの#include 文を出力します。<ファイル指定>は、ファイル名を" "または< >で括った形式で記述します。両者の相違は、#include 文の仕様の通りです。

複数ファイルを指定する場合は、スペースで区切ります。

■使用例

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

5.5 サーバ情報の定義文

このカテゴリには、以下の定義文があります。

- SVR_NAME{ … }; サーバ名称の定義
- SVR_ID{ … }; サーバ ID の定義
- SVR_VERSION{ … }; サーババージョンの定義
- SVR_NOINIT; サーバ初期化関数を生成しない
- SVR_NOSHUTDOWN; サーバ終了関数を生成しない
- SVR_NOSTUBTBL; サーバスタブ関数テーブルを生成しない
- SVR_STATIC{ … }; スタティックサーバの使用
- SVR_SECTION{ … }; サーバスタブに付与するセクション名の定義
- SVR_AUTH; サーバ ID・サーババージョンの与え方

5.5.1 SVR_NAME

■書式

```
SVR_NAME{<サーバ名>};
```

■解説

サーバ名称を定義します。

サーバ名称は、RPCGEN が生成するスタブ関数名などに利用されます。本定義文を省略することはできません。省略した場合 RPCGEN はエラーを出力し処理を中断します。

■使用例

```
SVR_NAME{EXAMPLE};
```

5.5.2 SVR_ID

■書式

```
SVR_ID{<ID 番号>};
```

■解説

サーバの ID を定義します。

ID 番号には、UINT32 で表現可能な 4 バイト整数定数、またはそれに置換される C 言語マクロ名のみを指定できます。C 言語マクロ名を記述する場合は、そのマクロは GLOBAL_INCFILE で指定されたファイルで定義されていなければなりません。

当然ながら、サーバ ID はシステム全体で一意でなければなりません。これは、HI7200/MP の仕様です。RPCGEN は、サーバ ID の重複を検出しません。

また、ID 番号は、<Config ファイル>_public.h へ以下の形式で出力されます。

```
#define  RPCSVR_ID_<サーバ名>  <ID 番号>
```

<サーバ名>は、SVR_NAME で指定したサーバ名です。

SVR_AUTH を省略した場合、本定義は省略できません。また、「5.5.8 SVR_AUTH」も参照してください。

■使用例

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

5.5.3 SVR_VERSION

■書式

```
SVR_VERSION{<バージョン>};
```

■解説

サーバのバージョンを定義します。

バージョンには、UINT32 で表現可能な 4 バイト整数定数、またはそれに置換される C 言語マクロ名のみを指定できます。C 言語マクロ名を記述する場合は、そのマクロは GLOBAL_INCFILE で指定されたファイルで定義されていなければなりません。

また、バージョンは、<Config ファイル>_public.h へ以下の形式で出力されます。

```
#define  RPCSVR_VERS_<サーバ名>  <バージョン>
```

SVR_AUTH を省略した場合、本定義は省略できません。また、「5.5.8 SVR_AUTH」も参照してください。

■使用例

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

5.5.4 SVR_NOINIT

■書式

```
SVR_NOINIT;
```

■解説

本指定を行うと、サーバ初期化関数を生成しません。

SVR_NOINIT は、アプリケーション側でサーバ初期化関数を用意する場合に指定してください。

また、「8.3 サーバ初期化関数」および「8.4 サーバスタブ関数テーブル」も参照してください。

■使用例

```
SVR_NOINIT;
```

5.5.5 SVR_NOSTUBTBL

■書式

```
SVR_NOSTUBTBL;
```

■解説

サーバスタブ関数テーブルとは、サーバ関数アドレスを保持するテーブルで、サーバ初期化関数が呼び出す rpc_start_server()または rpc_start_server_with_paramarea()で指定する rpc_server_info.ServerStubList となります。

本指定を行うと、サーバスタブ関数テーブルを生成しません。

SVR_NOSTUBTBL は、アプリケーション側でサーバスタブ関数テーブルを用意する場合に指定してください。

また、「8.4 サーバスタブ関数テーブル」も参照してください。

■使用例

```
SVR_NOSTUBTBL;
```

5.5.6 SVR_NOSHUTDOWN

■書式

```
SVR_NOSHUTDOWN;
```

■解説

本指定を行うと、サーバ終了関数を生成しません。

SVR_NOSHUTDOWN は、アプリケーション側でサーバ終了関数を用意する場合に指定してください。

RPCGEN が生成するサーバ終了関数では、rpc_stop_server()を呼び出してサーバを停止します。rpc_stop_server()には、サーバ停止時に実行するコールバック関数を指定できますが、RPCGEN が生成するサーバ終了関数では、常にコールバック関数を指定しません。コールバック関数を指定したい場合は、SVR_NOSHUTDOWN を指定し、ユーザ側でサーバ終了関数を実装してください。

また、「8.5 サーバ終了関数」も参照してください。

■使用例

```
SVR_NOSHUTDOWN;
```

5.5.7 SVR_STATIC

■書式

```
SVR_STATIC{<サイズ>△<セクション名>};
```

■解説

サーバをスタティックサーバとして登録します。本指定がない場合は、ダイナミックサーバとして登録します。

SVR_NOINIT と組み合わせて指定することはできません。

<サイズ>には、サーバパラメータ領域のサイズを UINT32 で表現できる整数定数で指定します。指定されたサイズは 4 の倍数に切り上げて扱われます。

<セクション名>には、サーバパラメータ領域に付与するセクション名を指定します。ただし、実際のセクション名は、指定したセクション名の先頭に'B'を付与した名称になります。

なお、この<セクション名>は、SVR_SECTION には影響されません。

サーバパラメータ領域は、<Config ファイル>_svr.c へ以下の形式で出力されます。

```
#pragma section <セクション名>
#pragma pack 4
static UINT8 ucServerArea_<サーバ名>[ ALIGNUP4(<サイズ>) ];
#pragma unpack
#pragma section
```

■注意事項

CPU コア間でキャッシュスヌープ機能を有さない SH2A-DUAL では、サーバパラメータ領域は非キャッシュ領域に配置する必要があります。指定するセクション名は、キャッシュ領域に配置するセクションと区別できるようにしてください。

■使用例

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

5.5.8 SVR_AUTH

■書式

```
SVR_AUTH;
```

■解説

サーバ ID およびサーババージョンを与える方法として、以下の 2 つの方式があります。

(1) モデル 1

SVR_AUTH 定義がない場合はモデル 1 となります。

このモデルでは、アプリケーションはサーバ ID とサーババージョンを意識しません。RPCGEN が生成したクライアント・サーバスタブ内で、使用するサーバ ID とサーババージョンを確定します。

クライアントスタブ関数は、オリジナルのサーバ関数と同じ API になります。また、以下の関数に渡されるサーバ ID とサーババージョンは無視されます。

- サーバ初期化関数
- サーバ終了関数
- クライアント初期化関数
- クライアント終了関数

RPCGEN は、サーバ ID とサーババージョンの定義を<Config ファイル>_public.h に出力します。RPCGEN が出力するクライアントスタブ関数およびサーバ・クライアントの初期化・終了関数では、<Config ファイル>_public.h をインクルードしてサーバ ID、サーババージョンを RPC ライブラリに渡します。

クライアント側とサーバ側で、世代の異なる RPCGEN 生成ファイルを使用した場合は、RPC ライブラリによって RPC コールがエラーとなります。

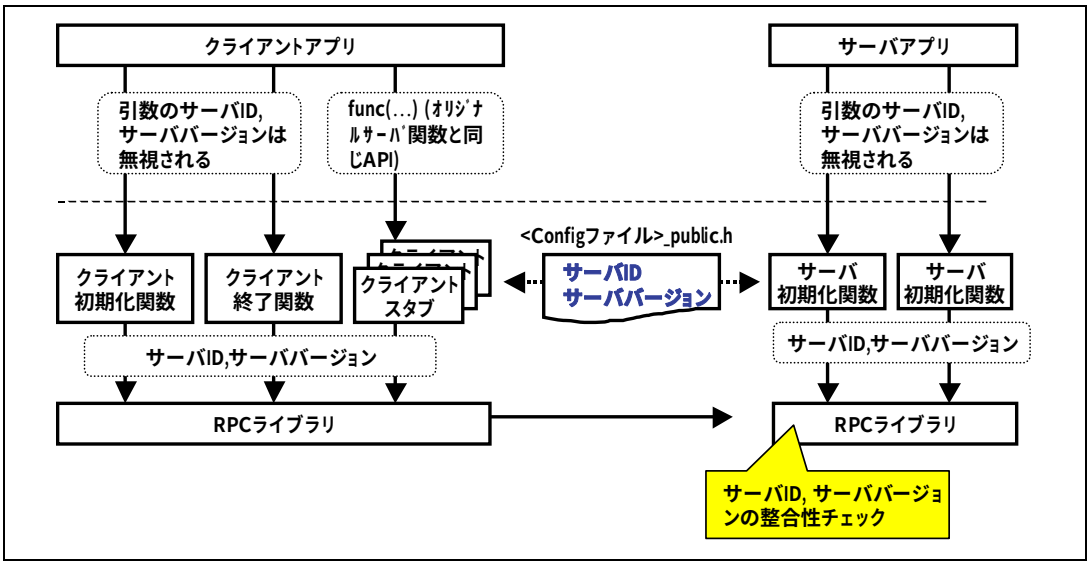

図 5.1 モデル 1

(2) モデル 2

SVR_AUTH 定義がある場合はモデル 2 となります。

このモデルでは、サーバ ID とサーババージョンは、アプリケーションから与えます。RPCGEN は、アプリケーションがサーバ ID とサーババージョンを取得できるようにするために、これらを定義した<Config ファイル>_pubic.h を出力します。

クライアントスタブ関数は、関数名が"rpcclnt_<サーバ名>_<サーバ関数名>となり、第 1 引数、第 2 引数はそれぞれサーバ ID とサーババージョンとなります。オリジナルのサーバ関数の引数は、第 3 引数以降となります。すなわち、クライアントスタブ関数は、オリジナルのサーバ関数とは異なる API を持ちます。オリジナルと同じ API で使用するには、クライアントスタブ関数のさらに上位にラッパ関数をユーザ側で実装する必要があります。

RPCGEN は、サーバ ID とサーババージョンの定義を<Config ファイル>_public.h に出力するので、クライアントスタブ関数およびサーバ初期化関数を呼び出すアプリケーションでは、<Config ファイル>_public.h をインクルードしてサーバ ID、サーババージョンを指定してください。

クライアント側とサーバ側で、世代の異なる<Cofig ファイル>_public.h をインクルードしていた場合は、RPC ライブラリによって RPC コールがエラーとなります。

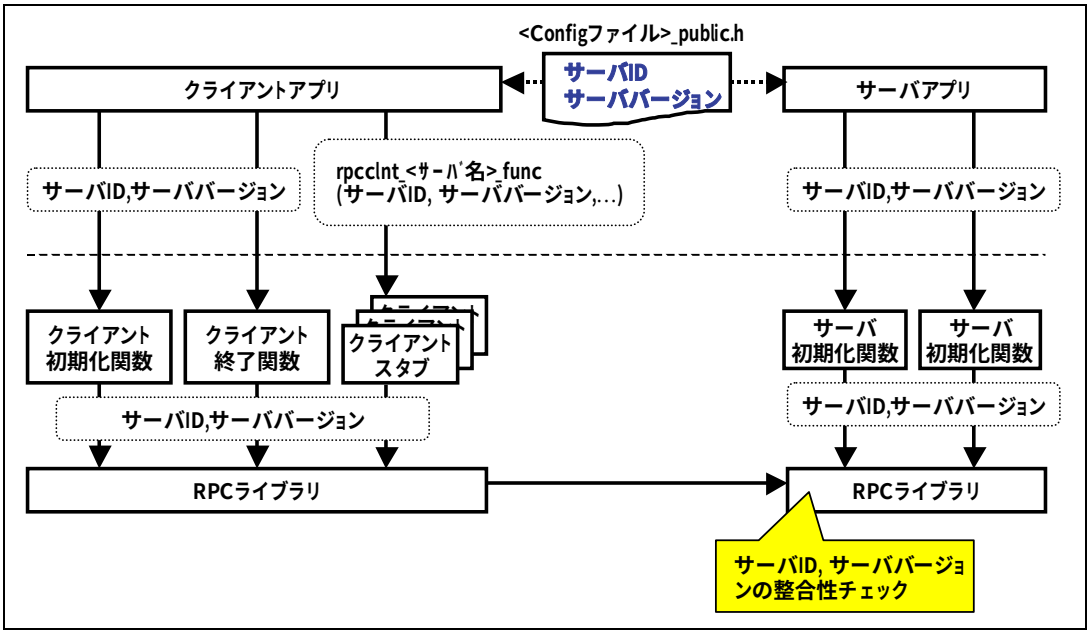

図 5.2　モデル 2

■使用例

```
SVR_AUTH;
```

5.5.9 SVR_SECTION

■書式

```
SVR_SECTION{<セクション名>};
```

■解説

サーバスタブを配置するセクションを指定します。本定義を省略した場合は、以下のセクション名となります。また、本定義を行った場合の実際のセクション名は、指定したセクション名の先頭に以下を付与した名称になります。

- プログラムセクション：'P'
- 定数セクション：'C'
- 未初期化データセクション：'B'
- 初期化データセクション：'D'

■使用例

```
SVR_SECTION{ C_EXAMPLE };
```

5.6 クライアント情報の定義文

このカテゴリには、以下の定義文があります。

- CLNT_NOINIT; クライアント初期化関数を生成しない
- CLNT_NOSHUTDOWN; クライアント終了関数を生成しない
- CLNT_CALLCHK; RPC コールのリターン値の保存
- CLNT_SECTION{ … }; クライアントスタブに付与するセクション名の定義

5.6.1 CLNT_NOINIT

■書式

```
CLNT_NOINIT;
```

■解説

RPCGEN は標準で、

```
INT32 rpcclnt_<サーバ名>_init(UINT32 __ulRPCServerID, UINT32 __ulRPCServerVersion );
```

というクライアント初期化関数を”Config ファイル名”_clnt.c に生成します。
CLNT_NOINIT を指定した場合は、これを生成しません。
CLNT_NOINIT は、ユーザ側でクライアント初期化関数を用意する場合に指定してください。
また、「8.6 クライアント初期化関数」も参照してください。

■使用例

```
CLNT_NOINIT;
```

5.6.2 CLNT_NOSHUTDOWN

■書式

```
CLNT_NOSHUTDOWN;
```

■解説

RPCGEN は、標準では

```
INT32 rpcclnt_<サーバ名>_shutdown(UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion );
```

というクライアント終了関数を<Config ファイル>_clnt.c に生成します。

CLNT_NOSHUTDOWN を指定した場合は、これを生成しません。

CLNT_NOSHUTDOWN は、アプリケーション側でクライアント終了関数を用意する場合に指定してください。

RPCGEN が生成するクライアント終了関数では、rpc_disconnect()を呼び出してサーバとの接続を切断します。rpc_disconnect()には、切断時に実行するコールバック関数を指定できますが、RPCGEN が生成するクライアント終了関数では、常にコールバック関数を指定しません。コールバック関数を指定したい場合は、CLNT_NOSHUTDOWN を指定し、ユーザ側でクライアント終了関数を実装してください。

なお、現在の HI7200/MP では、rpc_disconnect()でコールバック関数を指定しても無視される仕様となっています。

また、「8.7 クライアント終了関数」も参照してください。

■使用例

```
CLNT_NOSHUTDOWN;
```

5.6.3 CLNT_CALLCHK

■書式

```
CLNT_CALLCHK;
```

■解説

RPC コール(rpc_call()または rpc_call_copycbk())のリターン値を保持する目的で以下のコードを<Config ファイル>_clnt.c に生成し、*rpc_retval_adr()の API を<Config ファイル>_clnt.h へ出力します。

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

rpc_retval_adr()は、ユーザが実装する必要があります。
本指定を省略した場合は、以下のように RPC コールのリターン値が捨てられるコードが生成されます。

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

rpc_retval_adr()の実装方法については、「8.8 rpc_retval_adr()」を参照してください。

■使用例

```
CLNT_CALLCHK;
```

5.6.4 CLNT_SECTION

■書式

```
CLNT_SECTION{<セクション名>};
```

■解説

クライアントスタブを配置するセクションを指定します。本定義を省略した場合は、以下のセクション名となります。また、本定義を行った場合の実際のセクション名は、指定したセクション名の先頭に以下を付与した名称になります。

- プログラムセクション：'P'
- 定数セクション：'C'
- 未初期化データセクション：'B'
- 初期化データセクション：'D'

■使用例

```
CLNT_SECTION{ C_EXAMPLE };
```

5.7 サーバ関数の定義文

このカテゴリには、以下の定義文があります。

- RPC_FUNC{ … }; サーバ関数の定義

5.7.1 RPC_FUNC

■書式

```
RPC_FUNC
{
  <サーバ関数仕様定義>;
  ...
};
```

■解説

RPC_FUNC の{ }内には、各サーバ関数の定義文を記述します。
各サーバ関数の定義文は、「6. サーバ関数定義文の記述仕様」を参照してください。

■使用例

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

6. サーバ関数定義文の記述仕様

6.1 記述形式

サーバ関数定義は、以下のいずれかの形式で記述します。

(1) <リターン値仕様> <リターン変数名> = <関数名>(<引数部>, …)[< オプショナル指定>△…];

(2) <関数名>(<引数部>, …)[< オプショナル指定>△…];

(3) <リターン値拡張指令><リターン値仕様> <リターン変数名> = <関数名>(<引数部>, …) [< オプショナル指定>△…];

<引数部>、<オプショナル指定>、<リターン値拡張指令>には、RPCGEN のキーワードを[]で囲んで指定します。

例 1：

サーバ関数の仕様：`int func(int par);`

par を入力とし、int 型のリターン値を返す。

サーバ関数定義の記述：`int ret = func([IN DFLT]int par);`

解説：

- リターン値拡張指令：なし
- リターン値仕様：int ret =
- 関数名：func
- 引数部の引数指令：[IN DFLT]
- 引数部の引数仕様：int par
- オプショナル指定：なし

例 2：

サーバ関数の仕様：`void func(struct ST *ptr);`

ST 型の構造体へのポインタ ptr を入力とし、リターン値を返さない。

サーバ関数定義の記述：`func([IN PTR]struct ST *ptr)[UNACK];`

解説：

- リターン値拡張指令：なし
- リターン値仕様：なし
- 関数名：func
- 引数部の引数指令：[IN PTR]
- 引数部の引数仕様：struct ST *ptr
- オプショナル指定：[UNACK]

例 3：

サーバ関数の仕様：`double func(double inf);`

inf を入力とし、double 型のリターン値を返す。

サーバ関数定義の記述：`[RETEXT]double dret = func([IN DFLT]double inf);`

解説：

- リターン値拡張指令：[RETEXT]
- リターン値仕様：double dret =

- 関数名：func
- 引数部の引数指令：[IN DFLT]
- 引数部の引数仕様：double inf
- オプショナル指定：なし

それぞれの記述方法は、以下を参照してください。

- 関数型指令：「6.2 関数型指令の記述方法」
- リターン値仕様：「6.2 関数型指令の記述方法」
- 関数名：「6.3 関数名の記述方法」
- 引数部：「6.4 引数部」
- オプションナル指定：「6.7 オプショナル指定」

6.2 関数型指令の記述方法

6.2.1 リターン値を持つ関数の場合

■書式

<リターン値の型>△<リターン値格納変数>＝

■解説

リターン値格納変数とは、クライアントスタブおよびサーバスタブで、それぞれリターン値を格納するために使用するローカル変数です。関数引数と同じ変数を使用することもできます。

■記述例

サーバ関数の仕様：`int func(int par);`

par を入力とし、int 型のリターン値を返す。

サーバ関数定義の記述：**`int ret =`** `func([IN DFLT]int par);`

6.2.2 リターン値を持たない関数の場合

■解説

リターン値を持たない関数の場合は、リターン値仕様には何も記述してはなりません。

■記述例

サーバ関数の仕様：`void func(struct ST *ptr);`

入力はなく、*ptr に ST 型の構造体のリターンパラメータを返す。

サーバ関数定義の記述：`func([OUT PTR]struct ST *ptr);`

6.2.3　リターン値が整数型 4 バイトで表現できない場合(RETEXT)

■書式

[RETEXT]△<リターン値の型>△<リターン値格納変数>=

■解説

デフォルトでは、リターン値はサーバスタブで UINT32 型にキャストしてクライアントに転送され、クライアントスタブではそれをオリジナルの型にキャストしてリターンします。

リターン値が以下の型の場合は、上記枠組みでは対応できないため、RETEXT の指令が必要です。

- 64 ビット整数型
- 浮動小数点型
- 構造体型実体
- 共用体型実体

RETEXT を記述した場合、サーバからクライアントに IOVEC 構造体を利用してリターン値が返されるコードが生成されます。

■記述例

サーバ関数の仕様：`double func(int par);`

par を入力とし、double 型のリターン値を返す。

サーバ関数定義の記述：`[RETEXT]double ret = func([IN DFLT]int par);`

6.3　関数名の記述方法

■解説

関数名には、目的のサーバ関数名を記述してください。

■記述例

サーバ関数の仕様：`int func(int par);`

par を入力とし、int 型のリターン値を返す。

サーバ関数定義の記述：`int ret = func([IN DFLT]int par);`

6.4 引数部

引数部は、以下の形式で記述します。

[引数指令]<引数仕様>

引数指令は、後続の引数仕様に対する修飾です。引数指令は、以下の形式で記述します。

[<入出力種別>△<データ種別>]

入出力種別には、「6.5 入出力種別キーワード」に記載のキーワードを指定します。
データ種別には、「6.6 データ種別キーワード」に記載のキーワードを指定します。

引数仕様には、元となる関数の引数仕様を記述します。

入出力種別およびデータ種別の詳細は後述の通りですが、表 6.1にこれらキーワードの指定可能な組合せを示します。なお、REF と DESC を同時に指定することはできません。

表6.1　入出力種別とデータ種別の組み合わせ

入出力種別		データ種別			
	オプション	DFLT	STR	PTR	ARY
IN	無し	○	○	○	○
	REF	×	○	○	○
	DESC	○	○	○	○
OUT	無し	×	×	○	○
	REF	×	×	×	×
	DESC	×	×	×	×
INOUT	無し	×	×	○	○
	REF	×	×	×	×
	DESC	×	×	×	×

○：組合せ可能
×：組合せ不可(RPCGEN はエラーを通知して処理を中断します)

6.5 入出力種別キーワード

入出力種別キーワードには、引数の入出力属性を指定するキーワードとして、以下の 3 つがあります。

- IN　入力
- OUT　出力
- INOUT　入出力

また、上記 3 つのキーワードと組み合わせ可能なオプショナルなキーワードとして、以下があります。ただし、REF と DESC を同時に指定することはできません。

- REF　参照渡し
- DESC　クライアントからサーバに引数を受け渡さない

これらのキーワードは、後述のデータ種別キーワードと組み合わせて指定することで、後続に記述する引数仕様を修飾します。

6.5.1 IN(入力)

■解説

後続の引数が、サーバ関数への入力であることを定義します。
引数は、クライアントからサーバに転送され、サーバ関数に渡されます。

■使用例

サーバ関数の仕様：`int func(int par);`

par を入力とし、int 型のリターン値を返す。

サーバ関数定義の記述：`int ret = func([IN DFLT]int par);`

6.5.2 OUT(出力)

■解説

後続の引数が、サーバ関数からの出力であることを定義します。
サーバ関数が引数で指定された領域に出力したデータが、クライアントに返されます。
データ種別は、PTR, ARY とのみ組み合わせ可能です。

■使用例

サーバ関数の仕様：`void func(struct ST *ptr);`

入力はなく、*ptr に ST 型の構造体のリターンパラメータを返す。

サーバ関数定義の記述：`func([OUT PTR]struct ST *ptr);`

6.5.3 INOUT(入出力)

■解説

後続の引数が、サーバ関数への入力で、かつサーバ関数からの出力であることを定義します。
引数で指定された領域のデータは、クライアントからサーバに転送され、サーバ関数に渡されます。そして、サーバ関数が引数で指定された領域に出力したデータが、クライアントに返されます。
データ種別は、PTR, ARY とのみ組み合わせ可能です。

■使用例

サーバ関数の仕様：`int func(struct ST *ptr);`

`*ptr` を入力とし、同じ領域にリターンパラメータを返す。

サーバ関数定義の記述：`int ret = func([INOUT PTR]struct ST *ptr);`

6.5.4 REF(参照渡し)

■解説

後続の引数を、引数実体ではなく、そのアドレスのみをクライアントーサーバ間で受け渡します。REF 指定は、大きなサイズの引数を受け渡す場合に効果的です。
入出力種別は IN とのみ組み合わせ可能です。
データ種別は、PTR, STR および ARY とのみ組み合わせ可能です。
図 6.1に、REF 指定がない場合とある場合の違いを示します。

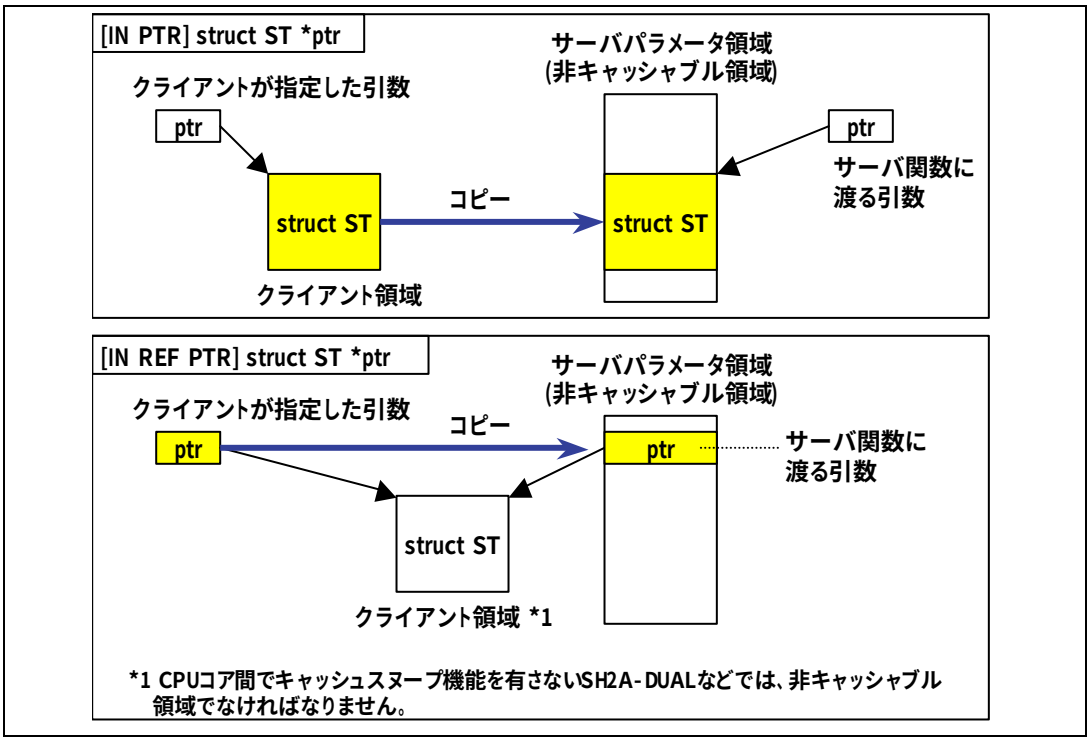

図 6.1　REF 指定

REF 指定がない場合は、PTR, STR または ARY で指定されたポインタ引数の指す内容が転送されます。転送されたポインタは、転送前のポインタとは別の領域を指すことになります。

REF 指定がある場合は、ポインタ引数そのものが転送されます。この場合は、転送前後のポインタ値は同じになります。

■注意事項

CPU コア間でキャッシュスヌープ機能を有さない SH2A-DUAL などでは、指定されるアドレスは非キャッシャブル領域でなければなりません。

■使用例

サーバ関数の仕様：`int func(struct ST *ptr);`

`*ptr` を入力として `func` に渡す。`ptr` は、必ず非キャッシャブル領域を指している。

サーバ関数定義の記述：`int ret = func([IN REF PTR]struct ST *ptr);`

6.5.5 DESC(クライアントからサーバに引数を受け渡さない)

■解説

後続の引数を、クライアントからサーバに受け渡しません。

入出力種別は IN とのみ組み合わせ可能です。

■使用例

サーバ関数の仕様：`int func(void);`

実際のサーバ関数には引数はないが、クライアント側のアプリケーションでは `func` を引数`(int par)` がある古い `API` で呼び出している。

サーバ関数定義の記述：`int ret = func([IN DESC PTR]int par);`

6.6 データ種別キーワード

データ種別キーワードには、以下の 4 種類があります。いずれかを指定します。

- DFLT　デフォルト
- PTR　ポインタ
- STR　文字列
- ARY　配列

なお、ARY の場合は別途 COUNT の引数部も指定する必要があります。

6.6.1 DFLT(デフォルト)

■入出力種別キーワードとの組み合わせ

IN のみと組み合わせ可能。

■解説

クライアントーサーバ間では、引数そのものが転送されます。
引数が以下のデータタイプの場合は、DFLT を指定できます。

- 整数型(signed short、unsigned short、signed long、unsigned long、signed int、unsigned int)
- 文字型(signed char、unsigned char)
- 実数型(float、double)
- 配列ではない構造体または共用体の実体

■記述例

サーバ関数の仕様：`int func(int par);`

par を入力とし、int 型のリターン値を返す。

サーバ関数定義の記述：`int ret = func([IN DFLT]int par);`

6.6.2 STR(文字列)

■入出力種別キーワードとの組み合わせ

IN のみと組み合わせ可能

■解説

クライアントからサーバへ、引数で指定された文字列そのものが転送されます。
引数が文字列型の場合に、STR を指定できます。
REF と組み合わせると、文字列そのものではなく文字列のポインタが転送されます。詳細は、「6.5.4 REF(参照渡し)」を参照してください。

■記述例

サーバ関数の仕様：`int func(const char *s);`

文字列 s を入力とし、int 型のリターン値を返す。

サーバ関数定義の記述：`int ret = func([IN STR]const char *s);`

6.6.3 PTR(ポインタ型)

■入出力種別キーワードとの組み合わせ

すべてと組み合わせ可能

■解説

クライアントーサーバ間で、指定されたポインタの指す内容が転送されます。

引数がポインタ型の場合に、PTR を指定できます。

入出力種別が IN の場合、REF と組み合わせると、ポインタの指す内容ではなくポインタ値が転送されます。詳細は、「6.5.4 REF(参照渡し)」を参照してください。

■記述例

サーバ関数の仕様：`void func(struct ST *ptr);`

入力はなく、`*ptr` に `ST` 型の構造体のリターンパラメータを返す。

サーバ関数定義の記述：`func([OUT PTR]struct ST *ptr);`

6.6.4 ARY(配列)

■入出力種別キーワードとの組み合わせ

すべてと組み合わせ可能

■解説

クライアントーサーバ間で、指定された配列の内容が転送されます。

引数が配列型の場合に、ARY を指定してください。後続の引数仕様には、配列の先頭アドレスを意味する引数を記述します。

ARY を指定する場合、ARY の直後にクライアントーサーバ間で転送する配列の要素数を指定するための COUNT 部の指定が必要です。詳細は、次項を参照してください。

入出力種別が IN の場合、REF と組み合わせると、配列実体ではなく配列へのポインタが転送されます。詳細は、「6.5.4 REF(参照渡し)」を参照してください。

■記述例

次項を参照してください。

6.6.5 COUNT(配列の要素数の指定)

■解説

ARY を指定した場合、COUNT によってクライアントーサーバ間で配列を転送する要素数を指定する必要があります。

(1) ARY の入出力種別が IN の場合

この場合、以下のように ARY を含む引数部に続いて、要素数を意味する COUNT 部をひとつ指定します。

```
<ARY を含む引数部>,<COUNT 部>
```

COUNT 部には、COUNT キーワードに続けてクライアントからサーバに受け渡す配列の要素数を式で指定します。

RPCGEN は、クライアントスタブコードを、図 6.2のように指定された式を用いて入力 IOVEC 構造体のサイズ情報を設定してから RPC コールするように生成します。このことを考慮して COUNT 部を指定してください。例えば、COUNT 部の式には、関数のリターン値を格納する変数を記述してはなりません。なお、図 6.2および図 6.3は後述の例 5 の場合のものです(SVR_AUTH 定義がない場合)。

```
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(...);
}
```

図 6.2　COUNT 部に対するクライアントスタブへの出力(IN)

```
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);
}
```

図 6.3　COUNT 部に対するサーバスタブへの出力(IN)

■記述例

例１：

サーバ関数の仕様：int func(struct ST *ptr);

要素数 10 の構造体配列のポインタ ptr を渡す。

サーバ関数定義の記述：int ret = func([IN ARY]struct ST *ptr, [**COUNT**]10);

例２：

サーバ関数の仕様：int func(struct ST *ptr, int count);

要素数が count の構造体配列のポインタ ptr を渡す。

サーバ関数定義の記述：int ret = func([IN ARY]struct ST *ptr, [**COUNT**]count,[IN DFLT]int count);

例３：

サーバ関数の仕様：int func(int count , struct ST *ptr);

要素数が count の構造体配列のポインタ ptr を渡す。

サーバ関数定義の記述：int ret = func([IN DFLT]int count, [IN ARY]struct ST *ptr, [**COUNT**]count);

例４：

サーバ関数の仕様：int func(struct ST *ptr, int *p_count);

要素数が*p_count の構造体配列のポインタ ptr を渡す。

サーバ関数定義の記述：int ret = func([IN ARY]struct ST *ptr, [**COUNT**]*p_count, [IN PTR]int *p_count);

例５：

サーバ関数の仕様：`int func(struct INF *inf, struct ST *ptr);`

`ptr` は関数に与える配列です。`inf` はその他の入力情報で、その中のメンバ `int count` が `ptr` の配列の要素数です。

サーバ関数定義の記述：`int ret = func([IN PTR]struct INF *inf, [IN ARY]struct ST *ptr, [COUNT]inf->count);`

(2) ARY の入出力種別が OUT の場合

この場合、以下のように ARY を含む引数部に続いて、要素数を意味する COUNT 部を二つ指定します。

```
<ARY を含む引数部>,<COUNT 部(1)> ,<COUNT 部(2)>
```

COUNT 部(1)には、COUNT キーワードに続けてクライアントが出力格納用に用意した配列の要素数を式で指定します。

COUNT 部(2)には、サーバが実際に出力する配列の要素数を式で指定します。実行時に出力する要素数は、必ず COUNT 部(1)で指定した要素数以下となるようにしなければなりません。

RPCGEN は、クライアントスタブコードを、図 6.4に示すように指定された式を用いて入力 IOVEC 構造体のサイズ情報を設定してから RPC コールするように生成します。また、サーバスタブコードを、図 6.5に示すようにサーバ関数コール後に指定された式を用いて出力 IOVEC 構造体のサイズ情報を設定するように生成します。このことを考慮して COUNT 部(1), (2)を指定してください。例えば、COUNT 部(1)の式には、関数のリターン値を格納する変数を記述してはなりません。なお、図 6.4および図 6.5は後述の例 8 の場合のものです(SVR_AUTH 定義がない場合)。

```
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(...);
}
```

図 6.4　COUNT 部(1)に対するクライアントスタブへの出力(OUT)

```
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);
}
```

図 6.5　COUNT 部(2)に対するサーバスタブへの出力(OUT)

■記述例

例１：

サーバ関数の仕様：`int func(struct ST *ptr);`

`ptr` は出力配列で、その要素数は `10` です。`func` は、必ず `10` 個の要素をすべて出力します。

サーバ関数定義の記述：

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

例２：

サーバ関数の仕様：`int func(struct ST *ptr, int *p_count);`

`ptr` は出力配列で、その要素数は `10` です。`func` のリターンパラメータ`*p_count` は出力要素数です。

サーバ関数定義の記述：

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

例３：

サーバ関数の仕様：int func(struct ST *ptr);

ptr は出力配列で、その要素数は 10 です。func のリターン値は出力要素数です。

サーバ関数定義の記述：int ret = func([OUT ARY]struct ST *ptr,

[**COUNT**]10, [**COUNT**]ret);

例４：

サーバ関数の仕様：int func(struct ST *ptr, int *p_count);

ptr は出力配列で、その要素数は*p_count です。func は、リターンパラメータ*p_count 要素数を出力します。

サーバ関数定義の記述：int ret = func([OUT ARY]struct ST *ptr,

[**COUNT**]*p_count, [**COUNT**]*p_count,[INOUT PTR]int *p_count);

例５：

サーバ関数の仕様：int func(struct ST *ptr, int count);

ptr は出力配列で、その要素数は count です。func は、常に count 個の要素を出力します。

サーバ関数定義の記述：int ret = func([OUT ARY]struct ST *ptr,

[**COUNT**]count, [**COUNT**]count,[IN DFLT]int count);

例６：

サーバ関数の仕様：int func(struct INF *inf, struct ST *ptr, int *p_count);

ptr は出力配列です。inf はその他の入力情報で、その中のメンバ int count が

ptr の配列の要素数です。func は、出力要素数を*p_count に出力します。

サーバ関数定義の記述：int ret = func([IN PTR]struct INF *inf, [OUT ARY]struct ST *ptr,

[**COUNT**]inf->count, [**COUNT**]*p_count, [OUT PTR]int *p_count);

例７：

サーバ関数の仕様：int func(struct INF *inf, struct ST *ptr);

ptr は出力配列です。inf はその他の入力情報で、その中のメンバ int count が

ptr の配列の要素数です。func は、常に inf->count の要素数を出力します。

サーバ関数定義の記述：int ret = func([IN PTR]struct INF *inf, [OUT ARY]struct ST *ptr,

[**COUNT**]inf->count, [**COUNT**]inf->count);

例８：

サーバ関数の仕様：int func(struct INF *inf, struct ST *ptr);

ptr は出力配列です。inf はその他の入力情報で、その中のメンバ int count が

ptr の配列の要素数です。リターン値が出力要素数です。

サーバ関数定義の記述：int ret = func([IN PTR]struct INF *inf, [OUT ARY]struct ST *ptr,

[**COUNT**]inf->count, [**COUNT**]ret);

(3) ARY の入出力種別が INOUT の場合

この場合、以下のように ARY を含む引数部に続いて、要素数を意味する COUNT 部を二つ指定します。

```
<ARY を含む引数部>,<COUNT 部(1)> ,<COUNT 部(2)>
```

COUNT 部(1)には、COUNT キーワードに続けてクライアントからサーバに受け渡す配列の要素数を式で指定します。

COUNT 部(2)には、サーバが実際に出力する配列の要素数を式で指定します。実行時に出力する要素数は、必ず COUNT 部(1)で指定した要素数以下となるようにしなければなりません。

RPCGEN は、図 6.6のように指定された式を用いて入力 IOVEC 構造体のサイズ情報を設定するように、クライアントスタブコードを生成します。また、図 6.7のように指定された式を用いて出力 IOVEC 構造体のサイズ情報を設定するように、サーバスタブコードを生成します。このことを考慮して COUNT 部(1), (2)を指定してください。例えば、COUNT 部(1)の式には、関数のリターン値を格納する変数を記述してはなりません。なお、図 6.6および図 6.7は後述の例８の場合のものです。(SVR_AUTH 定義がない場合)

```
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(...);
}
```

図 6.6　COUNT 部(1)に対するクライアントスタブへの出力(INOUT)

```
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);
}
```

図 6.7　COUNT 部(2)に対するサーバスタブへの出力(INOUT)

■記述例

例１：

サーバ関数の仕様：`int func(struct ST *ptr);`

`ptr` は入出力配列で、その要素数は `10` です。`func` は、必ず `10` 個の要素をすべて出力します。

サーバ関数定義の記述：

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

例２：

サーバ関数の仕様：`int func(struct ST *ptr, int *p_count);`

`ptr` は入出力配列で、その要素数は `10` です。`func` のリターンパラメータ `*p_count` は出力要素数です。

サーバ関数定義の記述：

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

例３：

サーバ関数の仕様：int func(struct ST *ptr);

ptr は入出力配列で、その要素数は 10 です。func のリターン値は出力要素数です。

サーバ関数定義の記述：int ret = func([INOUT ARY]struct ST *ptr,

[COUNT]10, [COUNT]ret);

例４：

サーバ関数の仕様：int func(struct ST *ptr, int *p_count);

ptr は入出力配列で、その要素数は*p_count です。func は、リターンパラメータ*p_count に要素数を出力します。

サーバ関数定義の記述：int ret = func([INOUT ARY]struct ST *ptr,

[COUNT]*p_count,[COUNT]*p_count,[INOUT PTR]int *p_count);

例５：

サーバ関数の仕様：int func(struct ST *ptr, int count);

ptr は入出力配列で、その要素数は count です。func は、常に count 個の要素を出力します。

サーバ関数定義の記述：int ret = func([INOUT ARY]struct ST *ptr,

[COUNT]count, [COUNT]count, [IN DFLT]int count);

例６：

サーバ関数の仕様：int func(struct INF *inf, struct ST *ptr, int *p_count);

ptr は入出力配列です。inf はその他の入力情報で、その中のメンバ int count が

ptr の配列の要素数です。func は、出力要素数を*p_count に出力します。

サーバ関数定義の記述：int ret = func([IN PTR]struct INF *inf, [INOUT ARY]struct ST *ptr,

[COUNT]inf->count, [COUNT]*p_count, [OUT PTR]int *p_count);

例７：

サーバ関数の仕様：int func(struct INF *inf, struct ST *ptr);

ptr は入出力配列です。inf はその他の入力情報で、その中のメンバ int count が

ptr の配列の要素数です。func は、常に inf->count の要素数を出力します。

サーバ関数定義の記述：int ret = func([IN PTR]struct INF *inf, [INOUT ARY]struct ST *ptr,

[COUNT]inf->count, [COUNT]inf->count);

例８：

サーバ関数の仕様：int func(struct INF *inf, struct ST *ptr);

ptr は入出力配列です。inf はその他の入力情報で、その中のメンバ int count が

ptr の配列の要素数です。リターン値が出力要素数です。

サーバ関数定義の記述：int ret = func([IN PTR]struct INF *inf, [INOUT ARY]struct ST *ptr,

[COUNT]inf->count, [COUNT]ret);

6.7　オプショナル指定

オプショナル指定のために、サーバ関数毎のクライアント・サーバスタブの出力制御を指定する以下のキーワードがあります。

- SVRSTUB　サーバスタブの指定
- SVRFUNC　サーバ関数の指定
- CLNTSTUB　クライアントスタブの指定
- UNACK　クライアントスタブ RPC コール ACK 指定
- CLNTCOPYCBK　rpc_call_copycbk()による RPC コール指定

オプショナルキーワードを複数指定することもできます。但し、同じキーワードを複数指定することは禁止です。また、矛盾する組合せの場合 RPCGEN はエラーを出力し処理を中断します。
複数のキーワードを指定する場合は、それらをカンマで区切ります。
表 6.2に、オプションキーワードの指定可能な組み合わせを示します。

表6.2　オプショナルキーワードの指定可能な組み合わせ

	SVRSTUB	SVRFUNC	CLNTSTUB	UNACK	CLNTCOPYCBK
SVRSTUB		×	○	○	○
SVRFUNC			○	○	○
CLNTSTUB				×	×
UNACK					○
CLNTCOPYCBK					

○：組合せ可能
×：組合せ不可(RPCGEN はエラーを通知し処理を中断します)

6.7.1　SVRSTUB(サーバスタブの指定)

■書式

```
[SVRSTUB] <サーバスタブ関数名>
```

■解説

ユーザ側で用意したサーバスタブを使用し、RPCGEN はサーバスタブを生成しません。

■記述例

サーバ関数の仕様：`int func(int par);`

par を入力とし、int 型のリターン値を返す。func のサーバスタブ func_svrstub はユーザ側で用意するので、RPCGEN では生成しない。

サーバ関数定義の記述：`int ret = func([IN DFLT]int par)` **`[SVRSTUB]func_svrstub`**`;`

6.7.2 SVRFUNC(サーバ関数の指定)

■書式

[SVRFUNC]<置換後の関数名>

■解説

サーバスタブが呼び出すサーバ関数を、指定の名称に置き換えます。
本指定は、クライアントが呼び出す関数名とサーバ関数名が異なる場合に指定します。
SVRSTUB と組み合わせて指定することはできません。

■記述例

サーバ関数の仕様：`int func(int par);`

par を入力とし、int 型のリターン値を返す。ただし、サーバ側の func の実体は、func_main という名称で存在する。

サーバ関数定義の記述：`int ret = func([IN DFLT]int par)[SVRFUNC]func_main;`

6.7.3 CLNTSTUB(クライアントスタブの指定)

■書式

[CLNTSTUB]

■解説

ユーザ側で用意したクライアントスタブを使用し、RPCGEN はクライアントスタブを生成しません。

■記述例

サーバ関数の仕様：`int func(int par);`

par を入力とし、int 型のリターン値を返す。クライアントスタブはユーザ側で用意するので、RPCGEN では生成しない。

サーバ関数定義の記述：`int ret = func([IN DFLT]int par)[CLNTSTUB];`

6.7.4 UNACK(非同期呼び出し指定)

■書式

```
[UNACK]
```

■解説

クライアントは、非同期モードで RPC コールします。本指定がない場合は、同期モードでコールします。

CLNTSTUB と組み合わせて指定することはできません。

■注意事項

本指定を行う場合、サーバ関数のリターン値型が void で、かつ引数の入出力種別は全て IN でなければなりません。

■記述例

サーバ関数の仕様：`void func(int par);`

par を入力とする。非同期モードでコールする。

サーバ関数定義の記述：`func([IN DFLT]int par)` **`[UNACK]`**`;`

6.7.5 CLNTCOPYCBK (rpc_call_copycbk()による RPC コール指定)

■書式

```
[CLNTCOPYCBK]<関数名 1> <関数名 2>
```

■解説

RPC コールを、rpc_call_copycbk()で行います。

関数名 1 および関数名 2 は、それぞれ rpc_call_copycbk()で指定するコピーコールバック関数名です。

2 つの関数指定は必須です。

指定する関数のプロトタイプは RPC 仕様に合致したもので、クライアントスタブからインクルードされるファイルでプロトタイプ宣言されている必要があります。

■記述例

サーバ関数の仕様：`int func(struct ST *ptr);`

ptr は出力配列で、その要素数は 10 です。func は、必ず 10 個の要素をすべて出力します。

クライアントからサーバへの転送には copy1()、サーバからクライアントへの転送には copy2()を使用します。

サーバ関数定義の記述：

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

7. RPCGEN が対応できないサーバ関数

本章では、RPCGEN が対応できないサーバ関数について解説します。本章に該当するサーバ関数がある場合は、SVRSTUB, CLNTSTUB 定義によって、RPCGEN にそのサーバ関数のサーバスタブ・クライアントスタブを生成させないように指定し、ユーザがサーバスタブ・クライアントスタブを実装してください。

7.1 引数

RPCGEN は、以下の引数を持つサーバ関数のクライアント・サーバスタブを正常に生成することはできません。

(1) サイズが定まらないデータ型の引数は扱えません

ただし、サイズが定まらない引数でも、それがポインタの場合は、REF 指定によってポインタそのものを受け渡すスタブコードを生成することは可能です。

■例

サーバ関数の仕様：`int func(void *par);`

サイズの定まらないデータへのポインタ `ptr` を入力とし、`int` 型のリターン値を返す。

サーバ関数定義の記述：

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

この場合は、クライアントスタブのコンパイル時にエラーが発生します。

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

この場合は、ポインタ `par` の値そのものがサーバに受け渡されるコードが生成されます。

(2) 多重ポインタは、初段のみを認識します

RPCGEN は、初段のポインタのみを認識します。

■例

サーバ関数仕様：int func(int **ptr);

ptr は、入力で、int 型データへのポインタへのポインタ。

サーバ関数定義： int ret = func([IN PTR] int **ptr);

この場合、RPCGEN は、*ptr をクライアントからサーバに転送するコードを生成します。*ptr は目的の int 型データへのポインタですが、そのポインタの指す内容(**ptr)は転送されません。

(3) 関数ポインタ

RPCGEN は、関数ポインタを引数に持つサーバ関数には対応しません。

7.2 リターン値

(1) ポインタはサーバ側のアドレスとなります

リターン値としてポインタ型を返す関数の場合、リターン値はサーバ側のアドレスとなります。RPCGEN は、そのポインタの指す内容をサーバからクライアントに転送するコードは生成しません。

■例

サーバ関数仕様：int *func(int par);
　リターン値は int 型データへのポインタ。
サーバ関数定義：int *ret = func([IN DFLT]int par);
この場合、ret はサーバ関数が返したサーバ側の int 型データへのポインタ値となります。

8. アプリケーションインタフェース

8.1 RPCGEN が生成するクライアントスタブ関数

クライアントスタブ関数は、クライアント側のアプリケーションが呼び出す関数です。

本節では、RPCGEN が生成するクライアントスタブ関数の仕様を示します。本仕様にしたがって、クライアントスタブ関数を呼び出してください。また、CLNTSTUB 指定によってクライアントスタブ関数を生成しない場合は、本仕様にしたがってユーザ側でクライアントスタブ関数を実装してください。

SVR_AUTH については、「5.5.8 SVR_AUTH」を参照してください。

(1) SVR_AUTH 定義がない場合

クライアントスタブ関数は、オリジナルのサーバ関数と同じ API となります。

RPCGEN は、クライアントスタブ関数実体を<Config ファイル>_clnt.c に出力しますが、プロトタイプ宣言は出力しません。必要に応じて、プロトタイプ宣言を行ったヘッダファイルを用意し、CLNT_INCFILE または GLOBAL_INCFILE 定義でそのヘッダファイルを指定してください。

■例

Config ファイルの内容：

```
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);
};
```

クライアントスタブ関数仕様：

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

(2) SVR_AUTH 定義がある場合

クライアントスタブ関数は、オリジナルのサーバ関数と異なる関数名となり、引数としてサーバ ID とサーババージョンが追加されます。RPCGEN は、クライアントスタブ関数実体を<Config ファイル>_clnt.c に、そのプロトタイプ宣言を<Config ファイル>_clnt.h に出力します。

クライアントスタブ関数名は、"rpcclnt_<サーバ名>_<サーバ関数名>"となります。

<サーバ名>は SVR_NAME で指定したサーバ名、<サーバ関数名>は RPC_FUNC 定義文中のサーバ関数定義文で指定したサーバ関数名です。

クライアントスタブ関数の第 1, 第 2 引数は、それぞれサーバ ID とサーババージョンとなります。引数名は、それぞれ"__ulID", "__ulVers"固定です。第 3 引数以降はサーバ関数の第 1 引数以降と同じになります。引数名は、サーバ関数定義文で指定したものと同じになります。

■例

Config ファイルの内容：

```
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);
};
```

クライアントスタブ関数仕様：

```
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 RPCGEN が生成するサーバスタブ関数

サーバスタブ関数のプロトタイプ宣言は<Config ファイル>_svr.h に、関数実体は<Config ファイル>_svr.c に出力されます。

サーバスタブ関数名は、"rpcsvr_<サーバ名>_<サーバ関数名>"となります。

<サーバ名>は、SVR_NAME で指定したサーバ名、<サーバ関数名>は RPC_FUNC 定義文中のサーバ関数定義文で指定したサーバ関数名です。

サーバスタブ関数は RPC 内部からコールされるため、通常はユーザは RPCGEN が出力するサーバスタブ関数の API を意識する必要はありませんが、SVRSTUB 指定によってサーバスタブ関数を生成しない場合は、本仕様にしたがってユーザ側でサーバスタブ関数を実装してください。

■例

Config ファイルの内容：

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

サーバ関数仕様：

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

8.3 サーバ初期化関数

サーバ初期化関数は、サーバ側のアプリケーションが呼び出す関数で、rpc_start_server()またはrpc_start_server_with_paramarea()によってサーバを開始します。

SVR_NOINIT 定義がない場合は、RPCGEN は"rpcsvr_<サーバ名>_init"という名称のサーバ初期化関数を生成します。サーバ初期化関数のプロトタイプ宣言は<Config ファイル>_svr.h に、関数実体は<Config ファイル>_svr.c に出力されます。

<サーバ名>は、SVR_NAME で指定したサーバ名です。

SVR_NOINIT 定義がある場合は、RPCGEN はサーバ初期化関数を生成しません。必要に応じて、ユーザがサーバ初期化関数を実装してください。

参考用に、以下に RPCGEN が生成するサーバ初期化関数の仕様を示します。

(1) C言語API

```
INT32 rpcsvr_<サーバ名>_init( rpc_svr_config *__config);
```

(2) リターン値

rpc_start_server()またはrpc_start_server_with_paramarea()のリターン値を返します。

(3) 引数：__config

rpc_svr_config構造体の定義は、以下の通りです。各メンバの意味は、RPC仕様で定義されているrpc_server_info構造体の同名メンバと同じです。

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

(4) 仕様

SVR_STATIC定義がない場合はrpc_start_server()、SVR_STATIC定義がある場合はrpc_start_server_with_paramarea()を用いてサーバを開始します。

ただし、SVR_AUTH定義がない場合は、__config. ulRPCServerIDおよび__config. ulRPCServerVersionで指定されたサーバIDおよびサーババージョンは無視されます。代わりに、<Configファイル>_public.hに出力されたRPCSVR_ID_<サーバ名>およびRPCSVR_VERS_<サーバ名>の各マクロを、それぞれサーバIDおよびサーババージョンとしてサーバを開始します。

8.4 サーバスタブ関数テーブル

サーバスタブ関数テーブルとは、サーバ関数アドレスを保持するテーブルで、サーバ初期化関数が呼び出す rpc_start_server()または rpc_start_server_with_paramarea()で指定する rpc_server_info.ServerStubList となります。

(1) SVR_NOSTUBTBL 指定なし、SVR_NOINIT 指定なし

RPCGEN は、サーバ初期化関数とサーバスタブ関数テーブルを生成します。

RPCGEN が生成するサーバスタブ関数テーブルは、RPCGEN が生成するサーバ初期化関数のみが参照します。このため、RPCGEN は、<Config ファイル>_svr.c にサーバスタブ関数テーブルを static データとして出力します。

(2) SVR_NOSTUBTBL 指定なし、SVR_NOINIT 指定あり

この場合、RPCGEN はサーバ初期化関数を生成しませんが、サーバスタブ関数テーブルは生成します。

RPCGEN は、以下の形式でサーバスタブ関数テーブルを<Config ファイル>_svr.c に出力します。アプリケーションで用意するサーバ初期化関数では、このテーブルを参照して rpc_start_server()または rpc_start_server_with_paramarea()を呼び出すようにしてください。

```
UINT32 (* const __rpcsvr_<サーバ名>_StubTable[])(rpc_server_stub_info *) =
{
    <サーバスタブ関数名>,
    <サーバスタブ関数名>,
};
```

<サーバ名>は、SVR_NAME で指定したサーバ名です。

(3) SVR_NOSTUBTBL 指定あり、SVR_NOINIT 指定なし

この場合、RPCGEN はサーバ初期化関数を生成しますが、サーバスタブ関数テーブルは生成しません。

RPCGEN は、アプリケーション側で以下のサーバスタブ関数テーブルが存在する前提で、これを外部参照するサーバ初期化関数を<Config ファイル>_svr.c に出力します。

```
UINT32 (* const __rpcsvr_<サーバ名>_StubTable[])(rpc_server_stub_info *) =
{
    <サーバスタブ関数名>,
    <サーバスタブ関数名>,
};
```

<サーバ名>は、SVR_NAME で指定したサーバ名です。

(4) SVR_NOSTUBTBL 指定あり、SVR_NOINIT 指定あり

この場合、RPCGEN はサーバ初期化関数、サーバスタブ関数テーブルを生成しません。アプリケーション側でそれぞれ用意する必要があります。

8.5 サーバ終了関数

サーバ終了関数は、サーバ側のアプリケーションが呼び出す関数で、rpc_stop_server()によってサーバを終了します。

SVR_NOSHUTDOWN 定義がない場合は、"rpcsvr_<サーバ名>_shutdown"という名称のサーバ終了関数が生成されます。サーバ終了関数のプロトタイプ宣言は<Config ファイル>_svr.h に、関数実体は<Config ファイル>_svr.c に出力されます。

<サーバ名>は、SVR_NAME で指定したサーバ名です。

SVR_NOSHUTDOWN 定義がある場合は、RPCGEN はサーバ終了関数を生成しません。必要に応じて、ユーザがサーバ終了関数を実装してください。

参考用に、以下に RPCGEN が生成するサーバ終了関数の仕様を示します。

(1) API

```
INT32 rpcsvr_<サーバ名>_shutdown(UINT32 __ulServerID, UINT32 __ulServerVersion);
```

(2) リターン値

rpc_stop_server()のリターン値を返します。

(3) 引数：__ulServerID、__ulServerVersion

それぞれ、サーバIDとサーババージョンです。

ただし、SVR_AUTH定義がない場合は、これらは無視されます。代わりに、<Configファイル>_public.hに出力されたRPCSVR_ID_<サーバ名>およびRPCSVR_VERS_<サーバ名>の各マクロを、それぞれサーバIDおよびサーババージョンとしてサーバを終了します。

(4) 仕様

rpc_stop_server ()を用いてサーバを停止します。この際、サーバ停止コールバック関数は指定しません。サーバ停止コールバック関数を使いたい場合は、SVR_NOSHUTDOWN定義を行い、ユーザ側でサーバ終了関数を実装してください。

8.6 クライアント初期化関数

クライアント初期化関数は、クライアント側のアプリケーションが呼び出す関数で、rpc_connect()によってサーバと接続します。

CLNT_NOINIT 定義がない場合は、RPCGEN は"rpcclnt_<サーバ名>_init"という名称のクライアント初期化関数を生成します。クライアント初期化関数のプロトタイプ宣言は<Config ファイル>_clnt.hに、関数実体は<Config ファイル>_clnt.c に出力されます。

<サーバ名>は、SVR_NAME で指定したサーバ名です。

CLNT_NOINIT 定義がある場合は、RPCGEN はクライアント初期化関数を生成しません。必要に応じて、ユーザがクライアント初期化関数を実装してください。

参考用に、以下に RPCGEN が生成するクライアント初期化関数の仕様を示します。

(1) API

```
INT32 rpcclnt_<サーバ名>_init(UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion);
```

(2) リターン値

rpc_connect()のリターン値を返します。

(3) 引数：__ulRPCServerID、__ulRPCServerVersion

それぞれ、サーバIDとサーババージョンです。

ただし、SVR_AUTH定義がない場合は、これらは無視されます。代わりに、<Configファイル>_public.hに出力されたRPCSVR_ID_<サーバ名>およびRPCSVR_VERS_<サーバ名>の各マクロを、それぞれサーバIDおよびサーババージョンとしてサーバと接続します。

(4) 仕様

rpc_connect ()を用いてサーバと接続します。

8.7 クライアント終了関数

クライアント終了関数は、クライアント側のアプリケーションが呼び出す関数で、rpc_disconnect()によってサーバとの接続を解除します。

CLNT_NOSHUTDOWN 定義がない場合は、RPCGEN は"rpcclnt_<サーバ名>_shutdown"という名称のクライアント終了関数を生成します。クライアント終了関数のプロトタイプ宣言は<Config ファイル>_clnt.h に、関数実体は<Config ファイル>_clnt.c に出力されます。

<サーバ名>は、SVR_NAME で指定したサーバ名です。

CLNT_NOSHUTDOWN 定義がある場合は、RPCGEN はクライアント終了関数を生成しません。必要に応じて、ユーザがクライアント終了関数を実装してください。

参考用に、以下に RPCGEN が生成するクライアント終了関数の仕様を示します。

(1) API

```
INT32 rpcclnt_<サーバ名>_shutdown(UINT32 __ulRPCServerID, UINT32
__ulRPCServerVersion);
```

(2) リターン値

rpc_disconnect()のリターン値を返します。

(3) 引数：__ulRPCServerID、__ulRPCServerVersion

それぞれ、サーバIDとサーババージョンです。

ただし、SVR_AUTH定義がない場合は、これらは無視されます。代わりに、<Configファイル>_public.hに出力されたRPCSVR_ID_<サーバ名>およびRPCSVR_VERS_<サーバ名>の各マクロを、それぞれサーバIDおよびサーババージョンとしてサーバとの接続を切断します。

(4) 仕様

rpc_disconnect ()を用いてサーバと接続を解除します。この際、サーバとの接続解除コールバック関数は指定しません。なお、現在のHI7200/MPでは、rpc_disconnect()は何も処理せずに終了する実装となっています。

8.8 rpc_retval_adr()

rpc_retval_adr()は、CLNT_CALLCHK(「5.6.3 CLNT_CALLCHK」参照)を指定した場合に、ユーザが用意する必要があるクライアント側の関数です。

以下の仕様にしたがって作成してください。

(1) API

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

(2) 仕様

呼び出したコンテキスト(タスク)に対応したリターン値格納アドレスを返す。

(3) HI7200/MP用の実装例

```
#define MAXTSKID 10  /* 最大タスクID */
static INT32 retval[MAXTSKID+1];  /* 各コンテキスト用のリターン値を格納する配列 */
                     /* [0] : 非タスク用 */
                     /* [1]～[MAXTSKID] : 当該タスクID用 */
INT32 *rpc_retval_adr(void)
{
  INT32 *retval_adr;
  ID    myid;

  if(sns_ctx() == TRUE)
  {
    /* 非タスクコンテキストの場合 */
    retval_adr = retval;
  }
  else
  {
    /* タスクコンテキストの場合 */
    get_tid(%myid);
    retval_adr = &retval[GET_LOCALID(myid)];
  }
  return retval_adr;
}
```

8.9 <Config ファイル>_public.h

このファイルには、以下のマクロ定義が出力されます。

```
#define RPCSVR_ID_<サーバ名>  <サーバ ID>
#define RPCSVR_VERS_<サーバ名>  <サーババージョン>
```

<サーバ名>、<サーバ ID>、<サーババージョン>は、それぞれ SVR_NAME、SVR_ID、SVR_VERSION で指定したサーバ名、サーバ ID、サーババージョンです。

本ファイルの位置付けは、SVR_AUTH 定義の有無によって異なります。詳細は、「5.5.8 SVR_AUTH」を参照してください。

8.10 RPCGEN 生成プログラムが使用するローカル変数名

RPCGEN は、クライアント・サーバスタブソース内で"__"(アンダースコア 2 つ)で始まる名称のローカル変数を使用します。

一方で、クライアント・サーバスタブでは、関数定義部で指定した引数名をそのまま使用します。このため、関数定義部で指定する引数名は、"__"で始まらないようにしてください。

9. 注意事項

(1) RPCGEN が生成したソースの動作確認は必須です

弊社は、RPCGEN が生成したプログラムの動作保証は致しません。RPCGEN が生成したプログラムの動作確認は、必ず実施してください。

特に、RPCGEN は処理が継続できないなどの致命的なエラーを検出した場合以外は、エラーを報告しないことに注意してください。

(2) RPCGEN が生成したファイルは修正しないでください

RPCGEN が生成したソースファイルが期待した内容となっていない場合でも、生成されたソースファイルを手動で修正することは推奨できません。これは、RPCGEN を再実行することで、ユーザが編集したファイルが上書きされてしまう可能性があるためです。

このような場合は、Config ファイルの指定を見直してください。

10. エラーメッセージ

RPCGEN は、処理が継続できないなどの致命的なエラーを検出した場合に限り、エラーメッセージを出力して終了します。言い換えると、以下の例のようなケースでは、RPCGEN はあたかも正常にコード生成を終了したかのように振る舞うことに注意してください。

■検出されないエラーケースの例

⇒は、現在の実装における振舞いです。この振舞いは、将来変更になる場合があります。

- 同じキーワードを複数回記述している。
  ⇒最初に見つかったキーワードを有効とします。
- 同じ名称のサーバ関数を複数回定義している。
  ⇒生成されたスタブのコンパイル時に、エラーが報告されます。
- 有効でないキーワードを指定している。
  ⇒単に無視されます。
- SVR_ID(サーバ ID)に、4 バイトを超える値を指定している。
  ⇒rpc_public.h をインクルードするファイルのコンパイル時に、エラーが報告されます。
- PTR 指定した引数の型がポインタ型でない。
  ⇒RPC コール時に、CPU 例外発生などの異常動作となります。

10.1 エラー形式

本章では、以下の形式で出力されるエラーメッセージとエラー内容を説明します。

```
<Config ファイル> : (E) <エラーメッセージ>
```

10.2 全般

`Cannot open configuration file.`
　　指定されたConfigファイルにアクセスできません。

`No <キーワード> definition.`
　　<キーワード>は省略できません。

`The <キーワード> directory doesn't exist.`
　　PUB_INCPATHなどのディレクトリパスを指定する<キーワード>で指定されたディレクトリが存在しません。

`Cannot create file. File: <ファイル>`
　　<ファイル>を生成できません。

`Cannot create temporary file.`
　　カレントディレクトリに中間ファイルを生成できません。

10.3 RPC_FUNC 以外の定義文のエラー

`Illegal <キーワード> definition.`
<キーワード>の引数が不正です。

`Cannot define both <キーワード 1> and <キーワード 2>.`
<キーワード1>と<キーワード2>を同時に指定することはできません。

10.4 RPC_FUNC 定義文のエラー

`[RPC_FUNC] Cannot define both <キーワード 1> and <キーワード 2> for "<関数名>".`
<関数名>の定義において、<キーワード1>と<キーワード2>を同時に指定することはできません。

`[RPC_FUNC] Illegal <キーワード> definition for "<関数名>".`
<関数名>の定義において、<キーワード>の引数が不正です。

`[RPC_FUNC] Illegal COUNT definition for "<関数名>".`
<関数名>の定義において、前方のARY定義に対するCOUNT定義が不正です。

11. 例

本章では、ユーザの理解を深めるために、Config ファイルの記述例と、出力ファイル例を掲載します。サーバ関数としては、以下の C 言語標準ライブラリ関数と HI7200/MP のサービスコールを題材としました。これは、多くのユーザが関数仕様を理解していると考えたためです。これらの関数を RPC 化すべき、という趣旨ではありません。

表11.1 例題とした関数

サーバ関数	備考
double atof(const char *nptr);	C 言語標準ライブラリ関数(stdlib.h)
int atoi(const char *nptr);	C 言語標準ライブラリ関数(stdlib.h), REF 指定
ER ref_tsk2(ID tskid, T_RTSK *pk_rtsk);	サービスコール ref_tsk
ER ref_sem2(ID semid, T_RSEM *pk_rsem);	サービスコール ref_sem

11.1 Config ファイル(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 */
```

ルネサスマイクロコンピュータ開発環境システム
ユーザーズマニュアル
スタブジェネレータ　V.1.00

発行年月日　2008 年 3 月 17 日　Rev.1.01
発　　行　株式会社ルネサス テクノロジ 営業統括部
〒100-0004 東京都千代田区大手町 2-6-2
編　　集　株式会社ルネサスソリューションズ
グローバルストラテジックコミュニケーション本部
カスタマサポート部

株式会社 ルネサス テクノロジ 営業統括部　〒100-0004 東京都千代田区大手町2-6-2 日本ビル

営業お問合せ窓口
株式会社ルネサス販売

http://www.renesas.com

本社	〒100-0004	千代田区大手町2-6-2 (日本ビル)	(03) 5201-5350
西東京支社	〒190-0023	立川市柴崎町2-2-23 (第二高島ビル)	(042) 524-8701
東北支社	〒980-0013	仙台市青葉区花京院1-1-20 (花京院スクエア)	(022) 221-1351
いわき支店	〒970-8026	いわき市平宇田町120番地ラトブ	(0246) 22-3222
茨城支店	〒312-0034	ひたちなか市堀口832-2 (日立システムプラザ勝田)	(029) 271-9411
新潟支店	〒950-0087	新潟市東大通1-4-2 (新潟三井物産ビル)	(025) 241-4361
松本支社	〒390-0815	松本市深志1-2-11 (昭和ビル)	(0263) 33-6622
中部支社	〒460-0008	名古屋市中区栄4-2-29 (名古屋広小路プレイス)	(052) 249-3330
関西支社	〒541-0044	大阪市中央区伏見町4-1-1 (明治安田生命大阪御堂筋ビル)	(06) 6233-9500
北陸支社	〒920-0031	金沢市広岡3-1-1 (金沢パークビル)	(076) 233-5980
鳥取支店	〒680-0822	鳥取市今町2-251 (日本生命鳥取駅前ビル)	(0857) 21-1915
広島支店	〒730-0036	広島市中区袋町5-25 (広島袋町ビルディング)	(082) 244-2570
九州支社	〒812-0011	福岡市博多区博多駅前2-17-1 (博多プレステージ)	(092) 481-7695

※営業お問い合わせ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：コンタクトセンタ　E-Mail: csc@renesas.com

Colophon 8.0

スタブジェネレータ
V.1.00
ユーザーズマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J2124-0101