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。

＝＝＝　必ずお読み下さい　＝＝＝

M32Rファミリ用 C/C++コンパイラパッケージ

V.5.01 Release 00

リリースノート 第5版

株式会社ルネサス ソリューションズ

2009年 1月 15日

この度はM32Rファミリ用C/C++コンパイラパッケージ(以下、CC32Rと略します) V.5.01 Release 00 を採用いただきまして誠に有難うございます。
本資料はCC32Rのインストール方法および電子マニュアルの補足等について説明します。
電子マニュアルの該当項目をご覧になる場合は、併せてこのリリースノートをご覧いただきますようお願い申し上げます。

※ 本資料記載内容は、特性改良などにより予告なしに変更することがあります。

1. 前バージョンからの変更点

前バージョンV.5.00 Release 00からの主な変更点です。

■ C99言語仕様の一部に対応（C/C++コンパイラ）

C99 言語仕様(ISO/IEC 9899:1999)のうち、次の３つの項目に対応しました。

- 要素指示
  構造体や配列の初期化式に対し、それがどの要素に対するものかを直接選択する機能です。
- _Pragma 演算子
  #pragma を関数形式マクロの呼び出しと同じ条件で記述できるようにした演算子です。
- 可変個引数マクロ
  関数形式マクロにおいて、引数の個数が決まっていないマクロを定義することができます。

■ libg32R（ライブラリジェネレータ）の格納場所変更

ライブラリジェネレータの格納ディレクトリを、lib32R から bin32R に移動しました。

■ 次の８つの問題点を解消しました

- inline関数の展開がある場合、分岐命令の飛び先がおかしくなるか、内部エラーが発生する問題（C/C++コンパイラ）
  [RENESAS TOOL NEWS 2007年04月16日：RSO-M3T-CC32R-070416D 掲載]
- asm関数を-csオプションでコンパイルすると、オブジェクトにasm関数の内容が出力されない問題（C/C++コンパイラ）
  [RENESAS TOOL NEWS 2007年03月16日：RSO-M3T-CC32R-070316D 掲載]
- 最適化有効時にポインタと整数0を!=演算子で比較した結果が常に真になる問題（C/C++コンパイラ）
  [RENESAS TOOL NEWS 2006年04月16日：RSO-M3T-CC32R-060416D 掲載]
- リロケータブルロードモジュールがリンカに再入力できない問題（C/C++コンパイラ）
  [RENESAS TOOL NEWS 2006年03月16日：RSO-M3T-CC32R-060316D 掲載]
- 右パラメータが符号なしの整数値であるシフト演算を含む整数の定数式がエラーになる問題（C/C++コンパイラ）
  [RENESAS TOOL NEWS 2006年03月01日：RSO-M3T-CC32R-060301D 掲載]
- クラステンプレートから配列オブジェクトの一部が定義できない問題（C/C++コンパイラ）
- long型の最小値を定数のマイナス1で乗算または剰余算を行うと、opt.exeのアプリケーションエラーが発生する問題（C/C++コンパイラ）
- 同じ共用体のメンバを、0x80000000～0xffffffffの範囲の定数でマスクする場合に、内部エラーが発生する問題（C/C++コンパイラ）

■ マニュアル正誤表

添付マニュアルの正誤表を示しますので、下記内容に訂正のうえご参照ください。

【C/C++コンパイラ編】

ページ	箇所	誤	正
CC32Rマニュアル -vi (6/432)	第７章 組み込み用アプリケーションの作成	7.4 HEW におけるスタートアップファイル start.ms について	7.4 High-performance Embedded Workshop におけるスタートアップファイル start.ms について

《次頁へ続く》

《次頁からの続き》

ページ	箇所	誤	正
CC32Rマニュアル-12 (26/432)	2.2 前バージョンとの互換性 ■-MAPオプションのエラー処理の緩和	cc32Rのリンクマップファイル出力オプション、-Mオプション（または、lnk32Rの-Mオプション）を指定していた場合、...	cc32Rのリンクマップファイル出力オプション、-MAPオプション（または、lnk32Rの-Mオプション）を指定していた場合、...
CC32Rマニュアル-14 (28/432)	3.1.3 コマンド行の記述規則 図3.1 cc32Rコマンドの入力書式	-warn_suppressed_nested_comment	-warn_suppress_nested_comment
CC32Rマニュアル-136 (150/432)	7.4 (タイトル)	7.4 HEW におけるスタートアップファイル start.ms について	7.4 High-performance Embedded Workshop におけるスタートアップファイル start.ms について
CC32Rマニュアル-136 (150/432)	7.4 HEW におけるスタートアップファイル start.ms について 本文	HEW でプロジェクトを新規作成する際に作成する、start.ms は、TM およびユーザーズマニュアルで使用しているstart.ms を、HEW 用に変更したものです。	High-performance Embedded Workshop でプロジェクトを新規作成する際に作成する、start.ms は、TM およびユーザーズマニュアルで使用しているstart.ms を、High-performance Embedded Workshop 用に変更したものです。
CC32Rマニュアル-136 (150/432)	7.4 HEW におけるスタートアップファイル start.ms について 本文	内容は基本的に同じですが、HEW からアセンブラas32R の-D オプションに次のシンボルを指定することで、start.ms のパラメータを操作するようになっています。	内容は基本的に同じですが、High-performance Embedded Workshop からアセンブラas32R の-D オプションに次のシンボルを指定することで、start.ms のパラメータを操作するようになっています。
CC32Rマニュアル-136 (150/432)	7.4 HEW におけるスタートアップファイル start.ms について 表7.3 (表のタイトル)	表7.3 HEW におけるスタートアップファイル start.ms のシンボルの意味	表 7.3 High-performance Embedded Workshop におけるスタートアップファイル start.ms のシンボルの意味
CC32Rマニュアル-136 (150/432)	7.4 HEW におけるスタートアップファイル start.ms について 表7.3 HEW におけるスタートアップファイル start.ms のシンボルの意味	HEWのプロジェクト作成ダイアログでの項目名	High-performance Embedded Workshop のプロジェクト作成ダイアログでの項目名
CC32Rマニュアル-299 (313/432)	10.2.8 レジスタ（Registers） ■ ANSI 規格 6.5.1 記憶クラス指定子 ＜register 宣言できるオブジェクトの数＞	記憶クラス指定子registerは無視されます。	記憶クラス指定子registerは認識されますが、全てのregister指定されたオブジェクトがレジスタに割り付けられるとは限りません。実際にどのオブジェクトをレジスタに割り付けるかは、コンパイラが判断します。 register宣言できるオブジェクト数に制限はありません。

2. インストールを始める前に

インストールを開始するにあたり、以下の点にご注意ください。

■ TMについて

V.5.00 Release 00 以降のCC32Rでは、旧統合環境であるTMは使用できません。

CC32R 用の統合化開発環境は、より高機能な High-performance Embedded Workshop(本製品に同梱)に移行しておりますので、こちらをご利用いただきますようお願いいたします。

[TM について]
http://japan.renesas.com/fmwk.jsp?cnt=tm_mid_level_landing.jsp&fp=/products/tools/ide/tm/

[High-performance Embedded Workshop について]
http://japan.renesas.com/fmwk.jsp?cnt=hew_mid_level_landing.jsp&fp=/products/tools/ide/hew/

High-performance Embedded Workshop への移行に関しては、弊社ホームページに掲載しております「TM→HEW V.4 移行の手引き」を参考にしてください。

[TM→HEW V.4 移行の手引き (PDF:368KB)]
http://japan.renesas.com/media/support/faqimage/products/mpumcu/tool/hew_106627_pdf.pdf

■ ライセンスIDについて

インストールの途中でライセンスIDを入力する必要がありますので、インストールを始める前にライセンスID証書によりライセンスIDをご確認ください。

なお、V.5.00 Release 00 は有償バージョンアップとなっています。本バージョンの CC32R は、V.4.30 Release 00 以前のライセンスIDではインストールできませんのでご注意ください。

■ 必要なメモリおよびHDDの容量

CC32Rを快適に使用するためには、128Mバイト以上のメモリと1Gバイト以上の空きハードディスク領域が必要です。

■ 動作環境

Windows 3.1およびWindows NT 3.5x以前のバージョンでは動作しません。

■ ファイル名、ディレクトリ名

インストール先のディレクトリ名や、コンパイラに指定するソースプログラムのファイル名やディレクトリ名は、次の注意事項に従って下さい。

- 漢字や全角文字を含むディレクトリ名、ファイル名は使用できません。
- ファイル名に使用するピリオド（.）は１つのみ使用可能です。
- ネットワークパス名は使用できません。ドライブ名に割り当ててご使用下さい。
- ショートカットは使用できません。
- 空白を含むディレクトリ名、ファイル名は使用できません。
- "..."表記を用いて２つよりも上のディレクトリを指定することはできません。
- パス指定を含めたファイル名の長さが128文字以上になるものは使用できません。

■ SQMlint（ MISRA C ルールチェッカ ）を導入される場合

SQMlintのインストールは、CC32Rのインストールの後に行ってください。
SQMlintの後にインストールされたCC32Rからは、SQMlintの機能を利用することはできません。

詳しくは、SQMlintに添付のマニュアルまたはリリースノートを参照ください。

■ インストールマネージャ(V.5.01以降)について

V.5.01 Release 00 以降、CC32R のインストールは、インストールマネージャを経由して行います。このため、V.5.00 Release 00 以前の CC32R とはインストール手順が少し異なりますので、3.1「インストール手順」をご確認ください。

インストールマネージャには、一つの PC に統合環境 High-performance Embedded Workshop を複数インストールさせる機能があり、これを用いて複数の High-performance Embedded Workshop それぞれにツール環境(コンパイラやデバッガなどを組み合わせた単位)を構築することができます。

詳しくは、インストールマネージャを初めて実行する時に表示されるヘルプファイルを参照ください。

■ ユーザ登録のお願い

製品のインストールが完了しましたら、ユーザ登録を行ってください。
ユーザ登録されませんと、バージョンアップ情報の提供や技術サポートなどのサービスを受けることができません。

- ユーザ登録方法
  インストーラの最後で、ユーザ情報を表示するツールが表示されます。
  このツールで、「ユーザ登録用紙を作成する」を押し、作成したテキストファイルのし、テキストファイルを作成し、このファイル内容をすべてコピー＆ペーストして以下の電子メールアドレス宛てに送付してください。

  ```
  regist_tool@renesas.com
  ```

  電子メールをご利用になれない場合は、`regist.txt` のファイル内容をプリントアウトしてファクシミリで送付ください。送付先はライセンス ID 証書に記載されています。
  （ファクシミリの場合、ユーザ登録の完了まで多少日数がかかることがあります。できるだけ電子メールをご利用ください。）

(株)ルネサス テクノロジの個人情報保護方針につきましては、ルネサステクノロジのホームページ「個人情報保護について」をご覧ください。
ホームページ：
`http://japan.renesas.com/fmwk.jsp?cnt=privacy.htm&fp=/privacy/&site=i`
ユーザ登録でご提供頂きました個人情報は、お客様のサポート活動に活用させて頂き、そのために必要な範囲で(株)ルネサス テクノロジ、およびその関係会社、ならびに特約店に、電子データ、書面により提供させて頂きますので、ご了承の程お願い申し上げます。なお、提供を希望されない場合は、提供を停止させて頂きますので、お問い合せ時にその旨ご連絡ください。その場合、サポート範囲が制約される場合がございます。

■ 環境変数の設定

次の環境変数は必ず設定して下さい。設定しない場合や不正なパスを指定した場合は、動作中にエラーが発生することがあります。（ただし、統合化環境High-performance Embedded WorkshopからCC32Rを動作させる場合は設定の必要はありません）

```
M32RBIN
M32RLIB
M32RINC
M32RTMP
```

■ ウイルスチェックプログラム

ウイルスチェックプログラムが常駐した状態でインストーラを起動すると正常に動作しない場合があります。その場合はウイルスチェックプログラムの常駐を解除してからインストーラを起動し直してください。

■ 制限事項および使用上のご注意について

本書には、リリースされるバージョンにより、「制限事項」や「使用上のご注意」の章を設けている場合があります。これらの内容は、CC32Rをご使用の前に必ず参照ください。これらには、マニュアル作成後に判明した制限事項を記載しております。

■ CC32R の最新情報を入手するには

以下のURLにホームページを公開しています。こちらでは、ルネサス開発環境に関する最新情報が掲載されていますのでご活用ください。

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

3. CC32Rのインストール

3.1 インストール手順

1. **次の**A.**または**B.**のいずれかの方法に従ってインストーラを起動してください。**

[A. **添付**CD**から製品をインストールする場合**]

CDをPCのCD-ROMドライブにセットしてください。自動的にインストーラ(インストールマネージャ)が起動します。
もし自動的に起動しない場合は、CDのルートディレクトリにある `hewinstman.exe` を直接実行してください。

[B. **オンラインバージョンアップの場合**]

ホームページに記載している方法でインストーラをダウンロードし、実行してください。インストーラ(インストールマネージャ)が起動します。

2. **次の**A.**または**B.**のいずれかの順番でインストールマネージャを操作します。**

[A. **初めて**High-performance Embedded Workshop**をインストールする場合**]

※ High-performance Embedded Workshopがインストール済みの場合 → 「B. **すでに**High-performance Embedded Workshop**がインストールされている場合**」へ。

1) 「インストールを開始する」ボタンを押してください。

【ユーザ情報の入力について】

最初に会社名や連絡先などのお客様の情報の入力画面が出ることがありますが、これはユーザ登録のために必要なファイルを作成するのに使用します。
※ お客様の個人情報の取り扱いについては、本書「第2章インストールを始める前に」の「■ ユーザ登録のお願い」にてご確認ください。

2) 「High-performance embedded workshopを初めてインストールする」を選択し、「次へ」を押してください。

3) 「インストール先の選択」画面にHigh-performance Embedded Workshopのインストール先が表示されます。通常はこのままで問題ありませんので、「次へ」を押してください。

【インストール先ディレクトリについて】

ここで表示されているディレクトリは、統合環境High-performance Embedded Workshopのものです。CC32R（C/C++コンパイラ等のクロスツール群）のインストール先は、この後に起動するインストーラの中で選択するようになっていますのでご注意ください。

4) インストールしたいソフトウェアをチェック状態にして、「インストール」を押してください。
デフォルト状態では、コンパイラ本体「M32Rツールチェイン V.X.XX Release xx」および「オートアップデートユーティリティ」がチェックされています。

【オートアップデートユーティリティとは】

定期的にルネサス開発環境のホームページにアクセスし、各開発環境の更新状況を確認するツールです。

5) 「アクティブなHigh-performance embedded workshopを新規にインストールしますか?」と表示されますが、通常は更新しても問題ありませんので、「はい」を選択してください。

選択したインストーラが順に起動します。(→3.へ)

[B. **すでに**High-performance Embedded Workshop**がインストール済みの場合**]

1) 「インストールを開始する」ボタンを押してください。

2) 「アクティブなHigh-performance embedded workshop環境にインストールする(推奨)」を選択し、「次へ」を押してください。

3) 現在インストールされている「ツール構成」が表示されます。「次へ」を押してください。

4) インストールしたいソフトウェアをチェック状態にして、「インストール」を押してください。
デフォルト状態では、コンパイラ本体「M32Rツールチェイン V.X.XX Release xx」および「オートアップデートユーティリティ」がチェックされています。

【オートアップデートユーティリティとは】
定期的にルネサス開発環境のホームページにアクセスし、各開発環境の更新状況を確認するツールです。

5) 「アクティブなHigh-performance embedded workshopを更新しますか?」と表示されますが、通常は更新しても問題ありませんので、「はい」を選択してください。

選択したインストーラが順に起動します。(→3.へ)

3. **表示される各インストーラのメッセージに従ってインストールを実行してください。**

【インストール先ディレクトリについて】
CC32R（C/C++コンパイラ等のクロスツール群）のインストール先は、ここで起動されるインストーラの中で指定します。デフォルトは `C:¥Renesas¥CC32R¥v501r00` です。
必要に応じて変更してください。

【サポート情報ツールについて】
すでに入力されているユーザ情報によって、ユーザ登録とサポートのための用紙を作成するツールです。「ユーザ登録用紙のファイルを作成」または「サポート連絡書のファイルを作成」を押し、テキストファイルを作成することができます。
※ お客様の個人情報の取り扱いについては、本書「第2章インストールを始める前に」の「■ ユーザ登録のお願い」にてご確認ください。

【スタートメニューの構成】
インストール後は、［スタート］→［プログラム(P)］→［Renesas］に次のフォルダおよびショートカットが登録されます。
High-performance Embedded Workshop
M32R Family C,C++ Compiler V.x.xx Release xx (xには数値またはバージョン表記が入ります。)
ルネサス開発環境 HomePage

4. **インストールマネージャに戻ります**
「終了」を押してください。

3.2 アンインストール方法

不要になったプログラムはPCから削除することができます。

(1) Windowsのタスクトレイに、AutoUpdateのアイコンがないかどうか確認してください。
存在していれば、AutoUpdateを右クリックで選択し、「終了(E)」を押して終了させてください。（終了させない場合、アンインストールの際に、再起動を促す表示が出る場合があります。）

(2) Windowsの「コントロールパネル」から、「プログラムの追加と削除」（または「アプリケーションの追加と削除」(**)）を開いてください。

(3) 「プログラムの追加と削除」のリストから、「`High-performance Embedded Workshop`」を選択し「変更と削除」（「削除」または「追加と削除」の場合もあります(**)）を押してください。
(**) Windowsによりボタンの名前が異なります

(4) アンインストールが行われます。
「完了しました」と表示されたら、「OK」を押してください。

3.3 環境設定

DOSプロンプト上でCC32Rを利用する場合は、表１の環境変数を設定してください。

表1　CC32Rの環境変数設定例

環境変数	設定例
M32RBIN	SET M32RBIN=C:¥Renesas¥cc32r¥v501r00¥bin32R
M32RLIB	SET M32RLIB=C:¥Renesas¥cc32r¥v501r00¥lib32R
M32RINC	SET M32RINC=C:¥Renesas¥cc32r¥v501r00¥inc32R
M32RTMP	SET M32RTMP=C:¥Renesas¥cc32r¥v501r00¥TMP
M32RKIN	SET M32RKIN=sjis
M32RKOUT	SET M32RKOUT=sjis
コマンドパス	PATH %M32RBIN%;%PATH%

※ C:¥Renesas¥cc32r¥v501r00**をインストール先ディレクトリに選択した場合**

- 設定例は、インストーラのデフォルトのインストール先ディレクトリ（C:¥Renesas¥cc32r¥v501r00）に製品をインストールした場合です。インストーラのデフォルトと異なるディレクトリに製品をインストールした場合は、実際のディレクトリにあった内容に変更してください。
- M32RKIN, M32RKOUTは日本語処理機能の文字コードを指定します。これらには、自動的にsjisが設定されますので、入力も出力もShift-JISを使用する場合はこれらの環境変数は設定の必要はありません。
- 統合化環境High-performance Embedded Workshopから利用する場合は、環境変数の設定は必要はありません。

3.4 電子マニュアルの参照方法

電子マニュアルを参照するには、Acrobat Reader（アドビシステムズ社）などのPDFファイル表示プログラムが必要です。必要に応じてインストールしてください。

- Acrobat Readerについて
次のアドビシステムズ社のホームページからダウンロードしてください。
Acrobat Readerの動作環境や最新情報なども掲載されています。
http://www.adobe.co.jp/ （日本）
http://www.adobe.com/ （米国）

電子マニュアルを参照するには次の方法があります。

(1) スタートメニューから電子マニュアルを開く
インストール後は、電子マニュアルはWindowsのスタートメニューに登録されます。
［スタート］→［(すべての)プログラム(P)］→［Renesas］→ ［M32R Family C,C++ Compiler V.x.xx Release xx］
(xにはバージョン番号等が入ります）から、目的の電子マニュアルを選択してください。

(2) 電子マニュアルファイルをダブルクリックして開く
デフォルトのインストール先（C:¥Renesas¥cc32r¥v501r00）でインストールした場合、電子マニュアルをC:¥Renesas¥cc32r¥v501r00¥manualの下にインストールします。これらのファイル（拡張子.pdf）をダブルクリックすると、Acrobat Readerでファイルを見ることができます。）

電子マニュアルのファイル名を表２に示します。

表2　電子マニュアルファイル一覧

言語	マニュアルタイトル	PDFファイル名
日本語	M32Rファミリ用 C/C++コンパイラパッケージ C/C++コンパイラユーザーズマニュアル	cc32ruj.pdf
	M32Rファミリ用 C/C++コンパイラパッケージ アセンブラユーザーズマニュアル	as32ruj.pdf

	マップビューワ ユーザーズマニュアル	mapuj.pdf
英語	M32R Family C/C++ Compiler Package C/C++ Compiler User's Manual	cc32rue.pdf
	M32R Family C/C++ Compiler Package Assembler User's Manual	as32rue.pdf
	Map Viewer User's Manual	mapue.pdf

【補足】（マップビューワのマニュアルについて）

マップビューワのマニュアルは、NC30WAとの組み合わせをベースに記述されていますが、CC32Rとの組み合わせでも同様に利用することができます。

4. 追加機能の説明

本バージョンで追加された、マニュアルに記載されていない機能を説明します。

4.1 要素指示

4.1.1 概要

構造体、共用体および配列の初期化式中に、要素指示を含めることができます。

要素指示は、初期化式{...}中の初期化式または定数式の前に記述し、どの要素(構造体,共用体のメンバ、配列要素)に対する初期化式(定数式)なのかを選択します。

要素指示の形式は、.メンバ名(構造体・共用体のメンバ参照)と[定数式](配列参照)の組み合わせに等号(=)を付けたものです。

要素指示の例:

```
.member =
[3] =
[3][5].member =
.member[4] =
```

初期化式がオブジェクトのどの要素に対応するかが明確になるので、複雑な入れ子構造を持った構造体や配列を、従来よりもわかりやすく初期化することができます。

4.1.2 文法

要素指示:

要素指示子... =

要素指示子:

.メンバ名 または [定数式]

要素指示は、ひとつまとは複数の要素指示子の並びに、等号(=)を付加したものです。要素指示子は、.メンバ名 または [定数式] のいずれかで、 .メンバ名 は構造体・共用体のメンバを、 [定数式] は配列の何番目の要素かを選択します。

4.1.3 詳細説明

{ と } で囲む初期化式であれば、その内部の要素ひとつずつに、要素指示を付けることができます。

例えば、初期化式が { 1, 2, 3 } のような形式である場合、1, 2, 3 のいずれか、または全部に要素指示を付加することができます。このうち、2と3にだけ要素指示をするなら、

{ 1, 要素指示 2, 要素指示 3 }

という記述ができます。

例1

配列の場合

```
int d[] = { 1, [1] = 2, [2] = 3 };
```

構造体の場合

```
struct { int a, b, c; } e = { 1, .b = 2, .c = 3 };
```

初期化式が階層構造を持つ場合、例えば { 1, { 2, 3 }, 4 } の中では、{ 2, 3 } はひとつの要素でもあるので、

```
{ 1, 要素指示 { 2, 3 }, 4 }
```

という記述ができます。ただしこの場合、{ 2, 3 }に対する要素指示は、2個以上メンバを持つ構造体またはサイズが2個以上の配列のいずれかを指している必要があります。

例2

```
struct { int a, b[2], c; } f = { 1, .b = { 2, 3 }, 4 };
```

同様の要素指示を、2, 3 に対して直接行う、

{ 1, { 要素指示 2, 要素指示 3 }, 4 }

あるいは

{ 1, 要素指示 2, 要素指示 3, 4 }

という記述も可能です。

例3

```
struct { int a, b[2], c; } g = { 1, { [0] = 2, [1] = 3 }, 4 };
```

例4

```
struct { int a, b[2], c; } h = { 1, .b[0] = 2, .b[1] = 3, 4 };
```

4.1.4 その他の記述例

例5 配列と構造体の組み合わせ

```
struct { int a, b[2], c; } m[] = { [1].a = -1, [2].b = {1,2}, [2].c = 3 };
```

各要素は、x[1].a = -1, x[2].b[0] = 1, x[2].b[1] = 2, x[2].c = 3 と初期化されます。初期化式にはx[2]までの初期化があるので、x[]の要素数は3となります。

例6 構造体メンバの順序に依存しない初期化

```
struct { int num1, num2, num3; } n = { .num3 = 1, .num1 = -1 };
```

各要素は、n.num1 = -1, n.num3 = 1 となります。n.num2 は初期化式が省略されているので0に初期化されます。

例7 初期化する最初の場所を要素指示子で選択

```
float p[9] = {0.001, 0.01, 0.1, [6] = 10, 100, 1000};
```

要素指示子により、p[6] が 10に初期化されます。それに続く要素指示子のない初期化式はp[6]の続きとして処理され、p[7] = 100, p[8] = 1000 と初期化されます。

例8 共用体メンバの初期化

```
struct w { int a, b, c; };
union { struct w s, t, u; } q = { .t = { 1, 2, 3 } };
```

初期化に対してどの共用体メンバを意識しているかを明確にできます。

4.1.5 注意事項

・要素指示が実際のメンバや配列と対応が取れない場合は、コンパイル時にエラーとなります。

例9 要素指示の失敗

```
struct { int a, b[2], c; } r = { 1, { .b[0] = 2, .b[1] = 3 }, 4 };
```

これは、.b[0] = が、r.bを表す { } の内側に記述してあるので、r.b の .b[0] メンバという

存在しないオブジェクトに対する要素指示とみなされるからです。

・要素指示子を使った結果、同じ要素に対する初期化式が2つ以上存在する場合、後方の初期化式が有効になります。

例10 初期化される領域の重複

```
float s[5] = {0.001, 0.01, 0.1, [2] = 10, 100, 1000};
```

s[2] には、0.1 と 10 の2つの初期化式がありますが、後方の10が有効になります。

4.2 _Pragma演算子

4.2.1 概要

#pragmaを関数形式マクロの呼び出しと同じ条件で記述できるようにした演算子です。

マクロ置換を使って任意の#pragma機能を制御したい場合に便利です。

4.2.2 文法

_Pragma演算子は次のような形式で、引数として文字列リテラルを1つ受け付けます。

```
_Pragma(文字列リテラル)
```

4.2.3 詳細説明

_Pragma演算子は、関数形式マクロと同様に処理され、引数の文字列リテラルをパラメータとする#pragmaに置換されます。

たとえば、`_Pragma("SECTION B B1")` は、#pragma SECTION B B1 と同じ効果を持ちます。

例1

```
_Pragma("INTERRUPT func")  /* #pragma INTERRUPT func として処理される */
void func(void) { }
```

なお、_Pragma演算子を行の途中に書いた場合は、置換後の #pragma の前後には改行を付加した状態として認識します。

例2

```
_Pragma("SECTION B B1") int v2;
```

→ 次の記述に置換されます:

```
#pragma SECTION B B1
int v2;
```

4.2.4 その他の記述例

例3　特定機能の#pragmaをマクロ関数化する。

型がtでアドレスがaの変数vを定義するマクロ:

```
#define MAKE_IOPORT(v,t,a) PRAGMA_OPR(ADDRESS v a) t v
#define PRAGMA_OPR(x) _Pragma(#x)
MAKE_IOPORT(p4, unsigned short, 0x123400);
```

→ MAKE_IOPORTマクロは、次の記述に置換されます:

```
#pragma ADDRESS p4 0x123400
unsigned short p4;
```

4.2.5 注意事項

・引数を省略したり、2つ以上の文字列を書くことはできません。

```
×  _Pragma()
```

× _Pragma("INTERRUPT" "func")
× _Pragma("ADDRESS", "a", 0x12340000)

・末尾にセミコロンは付けないでください。

× _Pragma("INTERRUPT func");

4.3 可変個引数マクロ

4.3.1 概要

関数形式マクロにおいて、引数の個数が決まっていないマクロを定義することができます。

4.3.2 文法

#define マクロ名(...) 展開後文字列

または

#define マクロ名(仮引数の並び, ...) 展開後文字列

4.3.3 詳細説明

関数形式のマクロ定義において、仮引数として省略形式 ... を使うことができます。省略形式のすぐ右側にはコンマや他の仮引数を記述することはできません。

実引数は、左から順に省略形式でない仮引数に取り込まれていき、取り込まれずに残った実引数は全て省略形式に取り込まれます。このとき、置換後文字列中に __VA_ARGS__ があれば、省略形式に取り込まれた内容をコンマ区切りを含めてそのまま展開します。

例1

```
#define err_printf(fmt, ...) fprintf(stderr, fmt, __VA_ARGS__)

err_printf("Error: %s in line %d¥n", file, line)
```

→ 次の記述に置換されます:

```
fprintf(stderr, "Error: %s in line %d¥n", file, line)
```

仮引数が省略形式だけの場合は、実引数は全て省略形式に取り込まれます。

例2

```
#define mac2(...) int __VA_ARGS__

mac2(v21, v22, v23, v24);
```

→ 次の記述に置換されます:

```
int v21, v22, v23, v24;
```

4.3.4 注意事項

・仮引数の右端以外は、省略形式 ... にすることはできません。

・省略形式の仮引数を持つマクロの置換後文字列以外には、__VA_ARGS__ を記述することはできません。

5. 制限事項

現在確認されているCC32Rの制限事項です。

■ 非常に大きな実行文(コードサイズで128Kバイト以上)を持つ制御文や繰り返し文を記述した場合（C/C＋＋コンパイラ）

★ RENESAS TOOL NEWS [2007年08月01日 : 070801/tn2] 掲載

非常に大きなメモリを必要とする実行文を持つ制御文(if文など)や繰り返し文(while文など)を記述したCソースコードをコンパイルすると、次のようなエラーが発生してコンパイルできない場合があります。

```
a132R: "内部ファイル名": error: 16-bit displacement overflow in operand 2
```

※ "内部ファイル名"はコンパイル時に内部で自動的に作成されるファイル名です。コンパイル対象のC/C++ソースファイル名ではありません。

発生条件

次の条件をすべて満たす場合に発生します。

(1) コードサイズが128Kバイト以上になる、規模の大きな実行文がある。

(2) 上記(1)を実行文とする次の(a)～(e)いずれかの文がある

(a) if文。ただし、上記(1)の実行文がif判定式で真の場合の実行文である。

(b) switch文。ただし、実行文の合計が128Kバイトを超える場合。

(c) while文

(d) do ～ while文

(e) for文

発生例

[ソースファイル例]

[sample1.c]

```
-------------------------------------------------------------------------
int ans1;
void
func1(int a)
{
   if (a) {                        /* 発生条件(2)(a) */
      ;
    /* この部分が128KBに達する */    /* 発生条件(1) */
      ;
   } else {
     ans1 = 1;
   }
}
-------------------------------------------------------------------------
```

[sample2.c]

```
-------------------------------------------------------------------------
int ans2;
void
func2(int a)
{
   switch (a) {                   /* 発生条件(2)(b) */
     case 0:
       ;
```

```
      /* この部分が128KBに達する */    /* 発生条件(1) */
        ;
        break;
      case 1:
        ans2 = 2;
        break;
    }
  }
  ------------------------------------------------------------------------
```

[sample3.c]

```
  ------------------------------------------------------------------------
  void
  func3(int a)
  {
     while (--a) {                    /* 発生条件(2)(c) */
        ;
      /* この部分が128KBに達する */    /* 発生条件(1) */
        ;
     }
  }
  ------------------------------------------------------------------------
```

回避策

次の(1),(2)いずれかの方法で回避できます。

(1) 当該実行文をelse側の実行文と入れ替える(if文の場合)

当該実行文がelse側の時は今回の問題は発生しないため、判定式の意味を逆にし、真と偽の実行文を入れ替えてください。

変更の結果、真の実行文が空の場合は、この対策は効果がありませんので、何らかの実行文を記述してください。

[ソースファイル sample1.c の回避例]

```
  ------------------------------------------------------------------------
  int ans1;
  void
  func1(int a)
  {
    if (!a) {          /* 判定文の意味を逆にする */
       ans1 = 1;                    /* 実行文を入れ替え */
    } else {
       ;
     /* この部分が128KBに達する */   /* 実行文を入れ替え */
       ;
    }
  }
  ------------------------------------------------------------------------
```

(2) 当該実行文を関数化する(どの制御文、繰り返し文でも有効)

当該実行文の一部または全部を、新規作成の関数に移動し、それを呼び出すように変更してください。

なお、新規作成した関数は、inline宣言をしないでください。

[ソースファイル sample3.c の回避例]

```
  ------------------------------------------------------------------------
  void
  sub_func3(void)          /* 新規作成 */
  {
       ;
```

```
    /* この部分が128KBに達する */
      ;
}
void
func3(int a)
{
    while (--a) {
      sub_func3();     /* 新規作成の関数を呼び出し */
    }
}
------------------------------------------------------------------------
```

■ 20000個を超える引数を指定する場合（cc32R コンパイルドライバ）

cc32Rドライバは、20000個を超える引数を指定することができません。

もしこの個数を超える引数が指定された場合、cc32Rにおいて次のようなエラーが発生します。

```
<command line>: error: too many arguments
```

オブジェクトの個数が2万個近くあるような、きわめて大規模なアプリケーションでは、これらを一度にリンクしたときに、リンク時にこの制限に該当することがあります。

なお、リンカlnk32Rでリンクを行っている場合には、このような制限はありません。

これを回避するには、一度に指定するオブジェクトの個数を減らす必要があります。
この一例としては、オブジェクトをいくつかのグループに分けて、グループ単位でオブジェクトをライブラリ（lib32Rを使用）化し、これらをプロジェクトを指定する代わりにリンク対象にするという方法があります。

■ 16Mバイトを超えるライブラリの作成について（ライブラリアン）

CC32Rでは、ライブラリファイルのサイズは16Mバイトに制限されています。

もしこのサイズを超えるライブラリを作成使用とした場合、ライブラリアンlib32Rまたはコンパイラcc32Rにおいて、次のようなエラーが発生します。

```
lib32R: error: overflow total module size (>16Mbytes)
```

このような場合は、複数のライブラリファイルに分けて、それらをリンク時に全て指定するようにしてください。

なお、同様の理由で、オブジェクトの中に16Mバイトを超えるサイズのファイルがある場合はライブラリは作成できません。この場合はリンカに直接指定してください。

■ 統合化環境High-performance Embedded WorkshopにおけるCPUの設定項目とスタートアップファイルについて （統合化環境）

ルネサス統合化環境High-performance Embedded Workshopで新規ワークスペース作成時に生成されるスタートアップファイルは、CPUに特化した設定を行っておりません。各CPUのユーザーズマニュアルを参照の上、必要な制御レジスタの設定処理を追加してください。

なお、新規ワークスペース作成時指定したCPUシリーズおよびタイプは、スタートアップファイルではなく、オプションの初期設定に用います。

■ CTOR,VTBL,COMMONセクションの変更について（C／C＋＋コンパイラ）

コンパイラがC++言語の処理の際に作成するCTOR,VTBL,COMMONの3つのセクションは、#pragma SECTION でも -Rオプションを用いても変更することはできません。

これらのセクションは、標準ライブラリやリンカで特別な認識を行う必要があり、変更できるようにするとその処理に支障が出るためです。

■ C++言語における#pragma INTERRUPT (INTF) の制限事項（C／C＋＋コンパイラ）

割り込み処理関数であることを指定する #pragma INTERRUPT（および、#pragma INTF）は C++言語上でに記述することができます。ただし、指定された関数はC言語関数でなければなりません。

このため、C++言語上では、#pragma INTERRUPT に指定したい関数は、extern "C" を用いて宣言しておく必要があります。

■ C++において240文字を越える識別子を宣言する場合（C／C＋＋コンパイラ）

C++言語で241文字以上の識別子(変数名、関数名やクラス名)を宣言した場合は、表３に示すように、C言語と処理が異なります。

表3　241文字以上の識別子を記述した場合の処理

識別子の種類	Cにおける処理	C++における処理
グローバル名	最初の240文字のみ用い241文字め以降を捨てます。	最初の240文字のみ用い241文字め以降を捨てます。
その他の識別子 (関数名、クラス名、構造体名、共用体名、メンバ名、引数名など)	最初の240文字のみ用い241文字め以降を捨てます。	241文字目以降も認識して区別しますが、内部で名前を記号化します。 このため、リンクエラーなどでは、関数名は元の名前で表示できません。

■ C++で #line に範囲外の値を指定した場合（C／C＋＋コンパイラ）

C++で #line に32768以上の値を指定した場合は、C言語と処理が異なります

【注意】
CC32Rでは、C,C++言語のいずれの場合でも、#line には32768以上の値を指定することはできません。記述しても生成コードには影響はありませんが、コンパイル時にエラーが発生したり、生成されたオブジェクトコードをデバッグに用いると、ソース表示などに問題が発生する場合があります。

1 〜 32767
正常に処理します。

32768〜4294967295
内部エラーが発生する場合があります。
```
internal error: BAD Location
internal error: unreachable logic
```

4294967296以上
エラーになります。
```
error: invalid line number
```

6. V.5.01 Release 00 使用上のご注意

本バージョンにおいて確認されている使用上の注意事項です。

■ 一部のfloat型演算用ランタイム演算ルーチンで、スタック使用量の値が不明になる問題（Cコンパイラ）

★ RENESAS TOOL NEWS [2008年10月16日 : 081016/tn7] **掲載**

該当製品に含まれるstk32R (スタック使用量計算ユーティリティ)でスタック使用量を計算する際、標準ライブラリに含まれる一部のランタイム演算ルーチンのスタック使用量が算出されません。

発生条件

次の(1)および(2)の全ての条件を満たす場合に発生します。

(1) 次の(a)～(d)のいずれかに該当する演算が含まれるプログラムを、-stack オプションを使用してコンパイルし、スタック使用量表示ファイル(拡張子が.stk)を生成している。

(a) float型を符号なし整数型に変換

(b) float型同士の減算

(c) 符号なし整数をfloat型に変換

(d) double型を符号なし整数型に変換

(2) stk32Rでスタック計算している。このとき次の(a)および(b)を共に満たす。

(a) (1)で生成されたスタック使用量表示ファイルを入力している。

(b) -lオプションで、標準ライブラリ用スタック使用量表示ファイル(m32RcR.stk, m32RcRM.stk, およびm32RcRL.stk) のいずれかを指定している。

発生例

[ソースファイル例]
[sample.c]

```
float f1,f2,f3;
unsigned u;
double d;
void func(void)
{
   u = f1;      /* 発生条件(1)(a) */
   f1 = f2 - f3; /* 発生条件(1)(b) */
   f2 = u;      /* 発生条件(1)(c) */
   u = d;       /* 発生条件(1)(d) */
}
```

[コマンドライン例]

```
% cc32R -stack -c sample.c
% stk32R -efunc -lm32RcR.stk sample.stk
注: “%” は、プロンプトを表します。
```

発生条件(1)の(a)～(d)に該当する場合に使用されるランタイム演算ルーチンのスタックサイズ情報が標準ライブラリ用スタック使用量表示ファイルに含まれていないため、スタックサイズが算出されず次のように画面上に表示されます。

```
*** Stack Size ***

          12 bytes
```

```
                    12 bytes + _100_Fdtou
                    12 bytes + _100_Futos
                    12 bytes + _100_Fsubs
                    12 bytes + _100_Fstou
-------------------------------------------------------------------
```

回避策

次の手順を実施して回避してください。

(1) m32RcR_lacked.zip ファイルを下記URLからダウンロードして解凍してください。
http://tool-support.renesas.com/jpn/toolnews/081016/m32RcR_lacked.zip

(2) 解凍されたm32RcR_lacked.stkファイルを環境変数M32RLIBが指すディレクトリに格納してください。

(3) stk32R のコマンドラインに `-lm32RcR_lacked.stk` を追加してください。

[コマンドライン例]

```
-------------------------------------------------------------------
% stk32R -efunc -lm32RcR.stk -lm32RcR_lacked.stk sample.stk
注:“%”は、プロンプトを表します。
-------------------------------------------------------------------
```

以下のように算出したスタックサイズが表示されます。

```
-------------------------------------------------------------------
  *** Stack Size ***

          24 bytes
-------------------------------------------------------------------
```

■ float型の乗算と減算からなる演算式に対し、不正なFMSUB命令を生成する問題（Cコンパイラ）

★ RENESAS TOOL NEWS [2008年10月16日 : 081016/tn6] **掲載**

FMSUB命令の使用を許可するオプション `-m32re5` および `-fminst` を使用し、最適化オプションを使用してコンパイルする場合、float型の乗算と減算からなる演算式に対し生成されたFMSUB命令のオペランドが正しくないことがあります。

発生条件

次の(1)～(4)の全ての条件を満たす場合に発生することがあります。

(1) コンパイル時に `-m32re5` および `-fminst` オプションを共に使用している。

(2) コンパイル時に使用している最適化オプションが次の(a)か(b)のいずれかに該当する。

(a) `-O`, `-O1`, `-O3`, `-O5`, `-O7`のいずれかを使用している。

(b) `-Ospace`, `-Otime`のいずれかを使用し、かつ、`-O0`, `-O2`, `-O4`, `-O6` のいずれも使用していない。

(3) 次の(a)と(b)からなるfloat型の演算がある。

(a) 2項ともにfloat型である乗算

(b) (a)の結果から別のfloat型の値を減算

(4) (3)(b)の演算は、一旦auto変数へ代入された(a)の結果を参照している。

発生例

[ソースファイル例]

[sample.c]

```
-------------------------------------------------------------------
 float A, B, C;
 float func(void)
 {
    float var;
    float answer;
    /* 途中省略 */
```

```
    var = A * B;        /* 発生条件(3)(a) */
    answer = var - C;   /* 発生条件(3)(b),(4) */
    /* 途中省略 */
}
```

上記のソースコード例の途中省略部分のコードによっては、今回の問題に該当しない場合があります。

[コマンドライン例]

```
% cc32R -c -O7 -m32re5 -fminst sample.c   (発生条件(1)(2))
```

注：“%”は、プロンプトを表します。

ソースコード例では、answerを算出する正しい式は A * B - C になりますが、今回の問題に該当すると、A - B * C を意味する ”FMSUB A,B,C ”という正しくない命令を生成します。

回避策

内部制御オプション -Qa-Xs16 を指定してコンパイルしてください。これにより、今回の問題に該当するFMSUB命令の生成を抑止できます。

[コマンドライン例]

```
% cc32R -c -O7 -m32re5 -fminst -Qa-Xs16 sample.c
```

注：“%”は、プロンプトを表します。

■ 関数形式マクロの実引数に記述した、同じ関数形式マクロが展開されない問題（Cコンパイラ）

★ RENESAS TOOL NEWS [2008年09月01日 : 080901/tn3] **掲載**

複数行に渡って記述された関数形式マクロの実引数に、同じ関数形式マクロを記述した場合、実引数に記述された関数形式マクロが展開されない場合があります。

発生条件

以下のそれぞれの条件(1)(2)を共に満たす場合、本問題が発生します。

(1) 関数形式マクロを複数行にわたって記述している。

(2) (1)項のマクロの実引数に、(1)項と同じ関数形式マクロを使用している。

(3) (2)項の関数形式マクロが行の先頭にある。あるいは、行の先頭から(2)項の関数形式マクロとの間に空白類文字およびコメントしかない。

発生例

[ソースファイル例]

[sample1.c]

```
#define SMP_MACRO(a,b) (a + b)
int var = SMP_MACRO(1,                <- 発生条件(1)項
          SMP_MACRO(2,3));           <- 発生条件(2)および(3)項
```

[マクロ置換結果(cc32R -E sample1.c の出力内容)]

```
#line 1 "sample1.c"

int var = (1 +     SMP_MACRO(2,3)) ; /* SMP_MACRO(2,3)が展開されない */
```

回避策

次の(1),(2)いずれかの方法で回避できます。

(1) 発生条件(3)項の関数形式マクロを記述した行と直前の行を¥ 記号で連結する

[ソースファイル sample1.cpp の回避例]

元の初期化式並びを含む宣言を、例えば、sample1init.cpp という新規のC++ソースファイルに移動してください。さらに、元の宣言からは初期化式を削除し、extern宣言を付けてください。

[sample1.c の変更例]

```
----------------------------------------------------------------
#define SMP_MACRO(a,b) (a + b)
int var = SMP_MACRO(1,        ¥  <- この行の末尾に ¥ 記号を記述する
         SMP_MACRO(2,3));
----------------------------------------------------------------
```

(2) 同一フォーマットで名前の異なるもうひとつの関数形式マクロを定義し、それを実引数で使用する

[sample1.c の変更例]

```
----------------------------------------------------------------
#define SMP_MACRO(a,b) (a + b)
#define SMP_SAME_MACRO(a,b) (a + b)    <- SMP_MACROと同一内容のマクロ
int var = SMP_MACRO(1,
         SMP_SAME_MACRO(2,3));       <- SMP_MACROの代わりに使用する
----------------------------------------------------------------
```

■ 複数行にまたがる //または/*形式のコメントと共に記述関数形式マクロがエラーになる問題（Cコンパイラ）

★ RENESAS TOOL NEWS [2008年07月01日 : 080701/tn1] **掲載**

関数形式マクロを複数行にわたり、//形式のコメントと共に記述している場合、コンパイル時に次のようなエラーメッセージを発生することがあります。

[**エラーメッセージ**]

```
--------------------------------------------------------------------------
   error: unterminated comment
--------------------------------------------------------------------------
```

発生条件

次の(1)～(4)の全ての条件を満たす場合に発生することがあります。

(1) 展開対象となる関数形式マクロを複数行にわたって記述している。

(2) 関数形式マクロの最終行に、//または/*形式で始まるコメントがある。

(3) (2)の行のひとつ前の行に//形式で始まるコメントがある。

(4) 当該Cソースファイルのサイズ(他のファイルをインクルードしている場合はインクルードファイルサイズを含む)が512バイトよりも大きい。

発生例

[ソースファイル例]

[sample1.c]

```
----------------------------------------------------------------
#define SAMPLE_MACRO(a,b,c)  (a + b + c)
   /* . . . (途中省略) . . . */
int func(void)
{
   return SAMPLE_MACRO( 1,
                    2,   // No.2    <-- 発生条件(1)(3)
                    3);  // No.3    <-- 発生条件(1)(2)
}
```

```
------------------------------------------------------------------
```

上記の例で省略している部分のプログラム内容によっては問題が発生しない場合があります。

回避策

//形式から始まるコメントのうち、発生条件(3)に該当するものを、/* . . . */ 形式のコメントに変更するか、または削除してください。

[ソースファイル sample1.cpp の回避例]

元の初期化式並びを含む宣言を、例えば、sample1init.cpp という新規のC++ソースファイルに移動してください。さらに、元の宣言からは初期化式を削除し、extern宣言を付けてください。

[sample1.c の変更例]

```
------------------------------------------------------------------
return SAMPLE_MACRO( 1,
                     2,   /* No.2 */  /* コメントの形式を変更 */
                     3);  // No.3
------------------------------------------------------------------
```

■ 1万個を越える初期化式を記述した場合にコンパイルできない問題（C/C＋＋コンパイラ）

★ RENESAS TOOL NEWS [2007年08月01日 : 070801/tn1] **掲載**

C/C++ソースコード中に、{ と } で囲まれた初期化式並びの中に、1万～5万程度以上の初期化式がある場合、CC32Rが次のようなメッセージのWindowsエラーを発生し、コンパイルできないことがあります。

[Windows**エラーのメッセージ**]

```
------------------------------------------------------------------------------
問題が発生したため、postpar.exe を終了します。 ご不便をおかけして申し訳ありません。
------------------------------------------------------------------------------
```

発生条件

C++言語、C言語で発生条件が異なります。

以下のそれぞれの条件(1)(2)を共に満たす場合、本問題が発生する場合があります。

1. C++**言語の場合**

(1) short型またはchar型変数（unsignedやsignedが付加される場合を含みます）の配列宣言に対し、初期化式を約1万～5万個以上含む初期化式並びを記述している。

(2) 上記（1）の初期化式並びの前後の行に、別の宣言や関数の記述がある。

2. C**言語の場合**

(1) 任意型の配列宣言に対し、キャスト付きの初期化式を約1万～5万個以上含む初期化式並びを記述している。

(2) 次の（a）（b）いずれかの条件を満たす

(a) 上記(1)の配列宣言において配列個数が省略されている。

(b) 上記(1)の初期化式並びの後(下方)の行に、別の宣言や関数の記述がある。

発生例

[ソースファイル例]

[sample1.cpp]

```
------------------------------------------------------------------
unsigned short table1[] = {       /* 発生条件2.1(1) */
   0,
   112,
   93,
        :
```

```
    /* 3万個、同様の記述が続く */
        :
};
int
check_table1(int index)          /* 発生条件2.1(2) */
{
    int ans;
    ans = table1[index];
    return ans;
}
```

[sample2.c]

```
int table2[] = {                /* 発生条件2.2(1),(2)(a) */
    (unsigned) 0,
    (unsigned) 100392,
    (unsigned) 33293,
        :
    /* 5万個、同様の記述が続く */
        :
};
```

[sample3.c]

```
int table3[50003] = {            /* 発生条件2.2(1) */
    (unsigned) 0,
    (unsigned) 100392,
    (unsigned) 33293,
        :
    /* 以下、5万個、同様の記述が続く */
        :
};
int
check_table3(int index)          /* 発生条件2.2(2)(b) */
{
    int ans;
    ans = table3[index];
    return ans;
}
```

[コマンドライン例]

```
% cc32R -S sample1.cpp
% cc32R -S sample2.c
% cc32R -S sample3.c
```

注:“%”は、プロンプトを表します。

回避策

次の(1),(2)いずれかの方法で回避できます。

(1) 該当する初期化式並びを含む宣言を別のC++ソースに移動する(C++言語の場合)

[ソースファイル sample1.cpp の回避例]

元の初期化式並びを含む宣言を、例えば、sample1init.cpp という新規のC++ソースファイルに移動してください。さらに、元の宣言からは初期化式を削除し、extern宣言を付けてください。

[sample1init.cpp (新規作成)]

```
unsigned short table1[] = {
  0,
  112,
  93,
      :
  /* 3万個、同様の記述が続く */
      :
};
```

[sample1.cpp (変更後)]

```
extern unsigned short table1[]; /* 初期化式のないextern宣言に変更 */
int
check_table1(int index)
{
    int ans;
    ans = table1[index];
    return ans;
}
```

---

(2) 該当初期化式のキャストを削除する(C言語の場合)

[ソースファイル sample2.c の回避例]

初期化式に記述したキャストを削除してください。

[sample2.c (変更後)]

```
int table2[] = {
       0,                      /* キャストを削除 */
    100392,                     /* (以下同様) */
    33293,
      :
  /* 5万個、同様の記述が続く */
      :
};
```

---

(3) 当該宣言の配列個数の記述とソースコード末尾への移動(C言語の場合)

上記回避策の(2)が使用できない場合の代替手段です。

なお、発生条件2.2(2)の(a)と(b)両方に該当する場合は、下記の sample2.c と sample3.c で示す対策を共に実施する必要があります。

[ソースファイル sample2.c の回避例]

配列 table2 の個数を省略せずに記述してください。

[sample2.c (変更後)]

```
int table2[50003] = {            /* 個数を記述するように変更 */
  (unsigned) 0,
  (unsigned) 100392,
  (unsigned) 33293,
      :
  /* 5万個、同様の記述が続く */
      :
};
```

[ソースファイル sample3.c の回避例]

配列 table3 の初期化つき宣言を関数の後(下方)の行に移動し、元の場所には初期化式並

びのない宣言を記述してください。

[sample3.c (変更後)]

```
-----------------------------------------------------------------------
int table3[50003];              /* 宣言のみを残す */
int
check_table3(int index)
{
   int ans;
   ans = table3[index];
   return ans;
}
int table3[50003] = {            /* ここに移動する */
   (unsigned) 0,
   (unsigned) 100392,
   (unsigned) 33293,
      :
  /* 5万個、同様の記述が続く */
      :
};
-----------------------------------------------------------------------
```

7. ソフトウェアー覧

インストール後に作成されるディレクトリとファイル（統合環境High-performance Embedded Workshopのフォルダを除く）を表４に示します。

表4　インストール後のディレクトリ・ファイル一覧

ディレクトリ名	ファイル名	説明
bin32R	cc32R.exe	コンパイルドライバ （V.3.00.03.001）
	as32R.exe	アセンブルドライバ （V.2.05.00.000）
	lnk32R.exe	リンカ （V.1.15.01.000）
	lib32R.exe	ライブラリアン （V.1.06.00.000）
	lmc32R.exe	ロードモジュールコンバータ （V.1.14.00.000）
	map32R.exe	マップジェネレータ （V.1.23.02.000）
	libg32R.exe	ライブラリジェネレータ （V.1.00.00.000）
	strip32R.exe	デバッグ情報除去ユーティリティ （V.1.02.00.000）
lib32R	cpre.exe	プリプロセッサ （V.2.09.01.000）
	cpfrt32R.exe	C++パーサ (V.1.03.00.000)
	cfrt.exe	パーザ （V.2.29.01.000）
	postpar.exe	ポストパーザ （V.1.04.02.000）
	opt.exe	グローバルオプティマイザ (V.1.29.01.000)
	cg32R.exe	コードジェネレータ （V.4.05.01.000）
	genstk.exe	スタックファイルジェネレータ (V.1.00.00.000)
	a032R.exe	マクロプロセッサ （V.1.02.00.000）
	a132R.exe	アセンブラ （V.4.06.01.000）
	alis32R.exe	リストプロセッサ （V.1.02.00.000）
	parafilt.exe	パラレルプロセッサ （V.1.01.00.000）
	plink32R.exe	プレリンカ (V.1.00.00.000)
	conv32R.exe	SYSROF→ELFコンバータ （V.1.02.01.001）
	cmerge.exe	Cソースマージプロセッサ （V.1.03.00.000）
	m32RcR.lib	Cライブラリ （スモールモデル）
	m32RcRM.lib	〃 （ミディアムモデル）
	m32RcRL.lib	〃 （ラージモデル）
	m32RcR.stk	〃 スタック使用量計算ファイル (m32RcR.lib)
	m32RcRM.stk	〃 〃 (m32RcRM.lib)
	m32RcRL.stk	〃 〃 (m32RcRL.lib)
inc32R	assert.h, ctype.h, errno.h, float.h, limits.h, locale.h, long64.h, math.h, mathf.h, setjmp.h, signal.h, stdarg.h, stddef.h, stdio.h, stdlib.h, string.h, time.h, cstddef, cstdio, cstdlib, exception, new, new_ecpp.h, new_std.h, stdexcept, typeinfo	C標準ヘッダ、C++用ヘッダ
inc32R¥sys	assert.h, ctype.h, errno.h, float.h, limits.h, locale.h, math.h, mathf.h, setjmp.h, signal.h, stdarg.h, stddef.h, stdio.h, stdlib.h, string.h, time.h	システム定義ヘッダ 【注1 】
inc32R¥com	ANSI_errno.h, def.h, SBPP	
UnSpt32R 【注2 】	abslist.exe	ABSリストユーティリティ （V.1.02.00.000）
	stk32R.exe	スタック計算ユーティリティ （V.1.02.01.000）
	license_j.txt	ユーティリティ取扱い条件 （日本語）
	license.txt	ユーティリティ取扱い条件 （英語）
	abslist_j.txt	ユーティリティマニュアル （日本語）
	stk_j.txt	〃 （ 〃 ）
	abslist.txt	ユーティリティマニュアル （ 英語 ）
	stk.txt	〃 （ 〃 ）

lib32R	lbg_cc32r.dep	Cライブラリ 再生成用構成ファイル
lib32R¥src	～.cpp ～.c ～.ms	
lib32R¥pack	～.pack	
support	mtoolspt.htm	ルネサス開発環境 ホームページへのリンク
support¥cc32r	userinfo.txt regist.txt	インストール情報
smp32R	start.ms	スタートアップ、低水準サンプル
bin	mapviewer.exe map_inspect.dll mapviewer.hlp mapviewer.cnt	マップビューワ （V.3.00.00） マップビューワ用DLL マップビューワ ヘルプファイル マップビューワ ヘルプ設定ファイル
manual	cc32ruj.pdf as32ruj.pdf cc32rue.pdf as32rue.pdf mapuj.pdf	ユーザーズマニュアルCコンパイラ編〔日本語〕 ユーザーズマニュアルアセンブラ編〔日本語〕 ユーザーズマニュアルCコンパイラ編〔英語〕 ユーザーズマニュアルアセンブラ編〔英語〕 マップビューワ マニュアル 〔日本語/英語 (自動選択)〕
	～.msf	マニュアル属性ファイル
hew	～.det ～.dll ～.tbp	High-performance Embedded Workshop設定ファイル
tmp	（（中身なし））	CC32R作業用一時ファイル格納ディレクトリ

||||||| ご注意 |||||||

【注1】 inc32R¥sysに含まれるシステム定義ヘッダはCC32Rが参照しますので、変更および削除しないでください。変更および削除した場合の動作は保証されません。

【注2】 UnSpt32Rディレクトリ内に含まれるプログラムは、ライセンス形態やサポート等の取り扱いが他のCC32Rの構成物とは異なっております。
ディレクトリ内のlicense.sjの内容をご確認ください。