

nesC 1.1 言語リファレンスマニュアル日本語訳

2003年5月(原文)
2005年8月24日(日本語版 v1.0)

原文: <http://www.tinyos.net/tinyos-1.x/doc/nesc/ref.pdf>
David Gay, Philip Levis, David Culler, Eric Brewer
住友精密工業株式会社 宮本 哲(日本語訳)

日本語訳文書ライセンス

本文書の原文は GNU Free Documentation License の条項の下で配布されています。本文書もこれに準じます。GNU Free Documentation License については本文書の最後に記載しています。

1. イントロダクション

nesC は TinyOS の構造概念と実行モデルを統合するようデザインされた C 言語の拡張である。TinyOS は非常に限られた資源(例えば 8K バイトのプログラム領域、512 バイトの RAM)を持つセンサネットワークノードのためにデザインされた、イベント駆動 OS である。TinyOS は nesC で実装しなおされている。本マニュアルは nesC の v1.1 について述べる。また v1.0 からの変更点は Section 3 にまとめる。

nesC の背景にある基本概念は以下のとおりである。

- ・ 構造 (construction) と構成物 (composition) の分離: プログラムはコンポーネント (component) で作られ、コンポーネントによりプログラム全体が形作られるよう組み立てられる (ワイアリングされる)。コンポーネントは 2 つのスコープを定義する。1 つはコンポーネントの規定部スコープ(コンポーネントのインタフェースインスタンスの名前を含む)であり、もう 1 つはコンポーネントの実装部スコープである。コンポーネントはタスク (task) という形によって内部的に並列動作性を持つ。制御の流れはコンポーネントのインタフェースを通じて他のコンポーネントへと流れる。これらの制御の流れはタスクやハードウェア割り込みから生じている。
- ・ 一連のインタフェース (interface) によるコンポーネントの動作の規定。インタフェースはコンポーネントによって提供 (provide) されるか利用 (use) される。提供インタフェースは、コンポーネントがそのユーザに提供する機能をあらわし、利用インタフェースはコンポーネントが処理を行うために必要とする機能をあらわしている。
- ・ インタフェースは双方向性を持つ: インタフェースはインタフェースの提供者が実装すべき関数群 (コマンド) とインタフェースの利用者が実装すべき関数群 (イベント) を示す。これにより一つのインタフェースでコンポーネント間の複雑なやりとりを表現することができる (例えば興味のあるイベントに登録しておけば、そのイベントが起こったときにコールバックされる)。TinyOS ではすべての時間のかかる処理 (例: パケット送信) はノンブロッキングであるため、このことは重要である。時間のかかる処理の完了はイベント (例: 送信完了) を通じて知らされる。インタフェースを決めてやることによって、コンポーネントが `sendDone` (送信完了) イベントを実装しないで `send` コマンドを呼び出せないようにすることができる。概してコマンドは呼び下げ、すなわちアプリケーションコンポーネントからハードウェアに近いところへ向かって順に呼ばれていく。一方、イベントは呼び上げられていく。いくつかの低レベルイベントはハードウェア割り込みに結びつけられている (どのように結びつけられるかは基本的にはシステム依存であるので、本リファレンスマニュアルではこれ以上は述べない)。
- ・ コンポーネントはインタフェースを通じて互いに静的にリンクされる。これは実行時の効率を向上させ、堅牢性を増し、プログラムのよりよい静的解析を可能とする。

- ・ nesC はコードがすべてコンパイラによって生成されるようにデザインされている。これによってよりよいコード生成と解析が可能となる。例として nesC のコンパイル時のデータ競合検出があげられる。
- ・ nesC の並列処理モデルは処理完了まで走りきる (run-to-completion) タスクと割り込みハンドラに基づいている。割り込みハンドラはタスクや他の割り込みハンドラに割り込むことができる。nesC コンパイラは割り込みハンドラが起こしうる潜在的なデータ競合を通知してくれる。

本ドキュメントは nesC のチュートリアルというよりはリファレンスマニュアルである。nesC のイントロダクションとしては TinyOS チュートリアルの方がわかりやすい。

本ドキュメントの残りの部分は次のように構成されている: Section 2 ではリファレンスマニュアルで用いられる表記について説明する。Section 3 では nesC 1.1 の新しい機能についてまとめる。Section 4, 5, 6, 7 では nesC のインタフェースとコンポーネントについて述べる。Section 8 では nesC の並列処理モデルとデータ競合検出について述べる。Section 9 では nesC のインタフェースとコンポーネントがどのようにしてアプリケーションへ組み上げられるかを説明する。Section 10 では nesC の残りのいろいろな特徴についてふれる。最後に Appendix A にて nesC の文法 (K&R の Appendix A の C 言語文法への拡張) の全仕様を定義する。Appendix B は本リファレンスマニュアルで使われる用語集である。

2. 表記

タイプライタ書体 (typewriter) は nesC のコードやファイル名に使われる。1 文字の斜体文字 (添字がつくこともある) は nesC の構成要素 (entities) を示す。例: “コンポーネント *K*”、“値 *v*”。

nesC の文法は ANSI C 文法の拡張である。ANSI C 文法については K&R の Appendix A に基づくものとする。そこで述べられている文法についてはここでは繰り返さない。斜体文字の語はその文字どおりの用語を表していないことを示し、タイプライタ書体の語や記号は文字どおりの用語である。いくつかの場面で ANSI C 文法規則を変えているが、これは次のように示す: *also* は既存の非用語に別の規則を加えたことを示し、*replaced by* は既存の非用語を置き換えたことを示す。

nesC の構文の説明は対応する文法ブロック (fragments) に沿って行う。これらのブロックにおいて、省略部分 (そこでの構文にあまり関係のない) をあらわすために... をいくつか使う。

いくつかの例で C99 標準 `inttypes.h` に定義されている `uint8_t` と `uint16_t` を使う。

3. 変更点

nesC 1.0 から 1.1 への変更点は以下のとおりである。

1. `atomic` 文。並列処理のデータ構造の実装を容易にする。また、新しい、コンパイル時データ競合検出機構はこの文を利用する。
2. コンパイル時データ競合検出により変数への二つの割り込みハンドラ、あるいは割り込みハンドラとタスクからの潜在的な同時アクセスが `warning` として知らされる。
3. 割り込みハンドラ内で安全に実行されるコマンドとイベントは記憶クラス指定子の `async` により明示的に示されなければならない。
4. コマンドやイベントを “fan-out” (訳注: 一つのインタフェースに複数のコンポーネントがワイヤリングされた状態で、扇のように広がる様子) で呼び出した結果は新しい型指定である結合関数により自動的に結合される。
5. `uniqueCount` は新しい「定数関数」で、`unique` の利用数を数える。
6. NES C プリプロセッサシンボルは言語バージョンを示す。nesC 1.1 では 110 である。

¹ TinyOS ディストリビューションと一緒に <http://webs.cs.berkeley.edu> から入手できる。

4. インタフェース

nesC におけるインタフェースは双方向性を持つ: インタフェースは二つのコンポーネント、提供コンポーネント (*provider*) と利用コンポーネント (*user*) 間の多機能チャンネルを規定する。インタフェースは、提供側によって実装されるコマンド (*command*) という関数群と、利用側によって実装されるイベント (*event*) と呼ばれる関数群を規定する。

本セクションではインタフェースがどのように規定されるかを説明し、Section 5 ではどのようにしてコンポーネントが提供・利用するインタフェースを規定するのかを説明する。さらに Section 6 ではコマンドやイベントがどのようにして C のコードから呼ばれたり、どのようにして C のコードで実装されるかについて述べ、Section 7 でどのようにしてコンポーネントのインタフェースがリンクされるかを説明する。

インタフェースは以下のように、インタフェース型によって規定される²。

```
nesC-file:
    includes-listopt interface
    ...

interface:
    interface identifier { declaration-list }

storage-class-specifier: also one of
    command event async
```

これはインタフェース型の識別子 (*identifier*) の宣言である。この識別子はグローバルなスコープを持ち、コンポーネントとインタフェース型のそれぞれの名前空間に属する。そのため全てのインタフェース型は他のインタフェース型やすべてのコンポーネントと区別できる名前を持ち、通常の C の宣言との衝突は起こらない。それぞれのインタフェース型は宣言リスト (*declaration-list*) 内の宣言のための別々のスコープを持つ。この宣言リストは記憶クラスの *command* か *event* のついた関数宣言から構成されていなければならない(そうでなければコンパイルエラーとなる)。また *async* をつけるとそのコマンドやイベントが割り込みハンドラ内で実行可能であることを示す。

インタフェースは *include-list* を通して C ファイルをインクルードすることもできる (Section 9 参照)。

簡単なインタフェースの例は以下のようなものである。

```
interface SendMsg {
    command result_t send(uint16_t address, uint8_t length, TOS_MsgPtr msg);
    event result_t sendDone(TOS_MsgPtr msg, result_t success);
}
```

SendMsg インタフェース型の提供側は *send* コマンドを実装しなければならない。一方 SendMsg インタフェース型の利用側は *sendDone* イベントを実装しなければならない。

² 訳注: 表記の項で述べられているように構文記法はK&Rに従う。添字の"opt"は省略可能であることを示す。also one ofはCでの構文に加えて、それに続く集合も含めることを示す。

5.コンポーネント仕様

nesC のコンポーネントはモジュール (*module*, Section6) がコンフィグレーション (*configuration*, Section7) のいずれかである:

nesC-file:

```
includes-listopt module  
includes-listopt configuration  
...
```

module:

```
module identifier specification module-implementation
```

configuration:

```
configuration identifier specification configuration-implementation
```

コンポーネントの名前は識別子 (*identifier*) によって指定される。この識別子はグローバルなスコープを持ちコンポーネントとインタフェース型の名前空間に属する。コンポーネントは 2 つのコンポーネント毎のスコープをもつ: C のグローバルスコープに入れ子される規定部スコープと、規定部スコープに入れ子される実装部スコープである。

コンポーネントは *includes-list* を通じて C ファイルをインクルードすることもできる (Section9 参照)。

規定部にはこのコンポーネントによって利用されるか提供されるかする規定要素 (インタフェースインスタンス、コマンド、イベント) を並べる。Section4 でみたように、コンポーネントは自身が提供するインタフェースのコマンドと、自身が利用するインタフェースのイベントを実装しなければならない。さらに自身が提供するコマンドとイベントを実装しなければならない。

概してコマンドはハードウェアコンポーネントの方へ呼び下げ、イベントはアプリケーションコンポーネントの方へ呼び上げる (これはアプリケーションコンポーネントが一番上にあるようなコンポーネント図で nesC アプリケーションをみた場合を想定している)。制御の流れはコンポーネントの規定要素を通じてのみ交わる。

それぞれの規定要素は名前 (インタフェースインスタンス名、コマンド名、イベント名) を持つ。これらの名前はコンポーネントの規定部毎のスコープの変数の名前空間に属する。

specification:

```
{ uses-provides-list }
```

uses-provides-list:

```
uses-provides  
uses-provides-list uses-provides
```

uses-provides:

```
uses specification-element-list  
provides specification-element-list
```

specification-element-list:

```
specification-element  
{ specification-elements }
```

specification-elements:
specification-element
specification-elements specification-element

コンポーネントの規定部内に複数の `uses` や `provided` ディレクティブが現れてよい。複数の利用・提供規定要素は { と } で囲んで一つのディレクティブにまとめることができる。例えば以下の二つの規定部は同じである:

```

module A1 {
    uses interface X;
    uses interface Y;
} ...

module A1 {
    uses {
        interface X;
        interface Y;
    }
}

```

インタフェースインスタンスは以下のように規定される:

specification-element:
 interface *renamed-identifier* *parameters_{opt}*
 ...

renamed-identifier:
identifier
identifier as identifier

interface-parameters:
 [*parameter-type-list*]

インタフェースインスタンスの宣言の完全な構文は `interface X as Y` であり、明示的に `Y` を実体の名前と指定している。 `interface X` という構文は `interface X as X` の短い書き方である。

インタフェースパラメータ (*interface-parameters*) が省略された場合、 `interface X as Y` はこのコンポーネントへの一つのインタフェースに対応した単一なインタフェースインスタンスを表す。インタフェースパラメータがある場合 (例: `interface SendMsg S[uint8_t id]`) は、パラメータ付のインタフェースインスタンスの宣言である。これはこのコンポーネントへの複数のインタフェースに相当し、それぞれが個別のパラメータ値の組を持っている (したがって `interface SendMsg S[uint8_t id]` は `SendMsg` 型の 256 個のインタフェースの宣言である)。パラメータの型は整数型でなければならない (ここでは `enum` は許されない)。

コマンドやイベントは記憶クラス指定子の `command` や `event` をつけた標準 C 関数宣言を含めて規定要素として直接 (component 規定部に) 置くことができる。

specification-element:
declaration
 ...

storage-class-specifier: also one of
command event async

宣言が `command` か `event` の 記憶クラスのついた関数宣言でない場合はコンパイルエラーとなる。インタフェース内において `async` はそのコマンドやイベントが割り込みハンドラ内で実行可能であることを示す。

インタフェースインスタンスについて、インタフェースパラメータが規定されていない場合はコマンド(イベント)は単一コマンド(単一イベント)であり、インタフェースパラメータが規定されている場合はパラメータ付コマンド(パラメータ付イベント)である。インタフェースパラメータは関数の通常のパラメータリストの前に置く。例: `command void send[uint8_t id](int x)`:

direct-declarator: also
direct-declarator interface-parameters (parameter-type-list)
...

インタフェースパラメータはコンポーネント規定部内のコマンドとイベントに関してのみ許され、インタフェース型内では許されないことに注意すること。
規定部全体の例を示す。

```
configuration GenericComm {
  provides {
    interface StdControl as Control;
    interface SendVarLenPacket;

    // The interface are parameterised by the active message id
    interface SendMsg[uint8_t id];
    interface ReceiveMsg[uint8_t id];
  }
  uses {
    // signaled after every send completion for components which wish to
    // retry failed sends
    event result_t sendDone();
  }
} ...
```

この例において、`GenericComm` は、

- `StdControl` 型の単一インタフェースインスタンス `Control` を提供する。
- `SendVarLenPacket` 型の単一インタフェースインスタンス `SendVarLenPacket` を提供する。
- `SendMsg` と `ReceiveMsg` インタフェース型のパラメータ付インスタンスを提供する。パラメータ付インスタンスはそれぞれ `SendMsg` と `ReceiveMsg` と名づけられている。
- `sendDone` イベントを使う。

コマンド(イベント) F がコンポーネント K の規定部にて提供されていることを「 K の提供コマンド(イベント) F_1 」と言う。同様にコマンド(イベント) F がコンポーネント K の規定部にて利用されていることを「 K の利用コマンド(イベント) F_1 」と言う。

コンポーネント K の提供インタフェースインスタンス X に含まれるコマンド F は「 K の提供コマンド $X.F_1$ 」と言う。 K の利用インタフェースインスタンス X に含まれるコマンド F は「 K の利用コマンド $X.F_1$ 」と言う。 K の提供インタフェースインスタンス X に含まれるイベント F は「 K の利用イベント $X.F_1$ 」言い、 K の利用インタフェースインスタンス X に含まれるイベント F は「 K の提供イベント $X.F_1$ 」言う(イ

ンタフェースの双方向という性質によりイベントに対する利用と提供が逆になることに注意すること)。

利用・提供の区別が重要でない場合には簡単に「*K*のコマンドあるいはイベント」とも言う。*K*のコマンドあるいはイベントはパラメータ付かもしれないし、単一であるかもしれない。それは対応する規定要素がパラメータ付か単一であるかによるのだ。

6. モジュール

モジュールは C でコンポーネントの規定を実装する。

module-implementation:
implementation { *translation-unit* }

翻訳単位 (*translation-unit*) は C の宣言と定義のリストである。

モジュールの翻訳単位のトップレベルの宣言はそのモジュールのコンポーネントの実装部スコープに属する。この宣言は無限の大きさを持ち、標準 C の宣言や定義、タスク宣言や定義、コマンドやイベントの実装のいずれをも含むことができる。

6.1 モジュールの *specification* の実装

translation-unit はそのモジュールの全ての提供コマンド(イベント) を実装しなければならない(すなわちすべての直接に提供されるコマンドとイベント、全ての提供インタフェースのコマンド、全ての利用インタフェースのイベント)。モジュールは全ての自身のコマンドを呼んだり、全ての自身のイベントをシグナルしたりすることができる。

これらコマンドとイベントの実装は以下の C 構文拡張にて規定される。

storage-class-specifier: also one of
command event async

declaration-specifiers: also
default *declaration-specifiers*

direct-declarator: also
identifier . identifier
direct-declarator interface-parameters (parameter-type-list)

単一のコマンドやイベント の実装は記憶クラス `command` あるいは `event` を付した の C 関数定義を持つ (*direct-declarator* の拡張により関数名に `.` を使うことができる)。これに加えて の宣言に `async` が含まれるならば `async` キーワードをつけなければならない。

例えば `SendMsg` 型のインタフェース `Send` を提供するモジュールにおいて:

```
command result_t Send.send(uint16_t address, uint8_t length,  
                           TOS_MsgPtr msg) {  
    ...  
    return SUCCESS;  
}
```

インタフェースパラメータ *P* を持つパラメータ付コマンドあるいはイベント の実装は記憶クラスの `command` が `event` をつけた の C 関数定義の構文をしていて、そこでは関数の通常のパラメータリストの前に角括弧で囲まれたパラメータ *P* が置かれている(コンポーネント規定部内のパラメータ付コマンドやイベントの構文と同じである)。このインタフェースパラメータ宣言 *P* は の関数パラメータスコープに属し通常の間数パラメータと同じ範囲である。例えば、`SendMsg` タイプのインタフェース `Send[uint8_t id]` を提供するモジュールでは:

```

command result_t Send.send[uint8_t id](uint16_t address, uint8_t length,
                                       TOS_MsgPtr msg) {
    ...
    return SUCCESS;
}

```

以下のような場合にはコンパイル時にエラーとなる:

- ・ 提供コマンドやイベントの実装がない
- ・ 与えられたモジュールの規定とマッチしない型名、インタフェースパラメータ、コマンドやイベントの `async` の有無

6.2 コマンドの呼び出しとイベントのシグナル

コマンドの呼び出しやイベントのシグナルのために以下の C 構文拡張が使われる:

postfix-expression:

```

postfix-expression [ argument-expression-list ]
call-kindopt primary ( argument-expression-listopt )
...

```

call-kind: one of

```

call signal post

```

単一コマンド は `call (...)` で呼び出され、単一イベント は `signal (...)` でシグナルされる。例えば `SendMsg` 型の `Send` インタフェースを利用するモジュールでは、`call Send.send(1, sizeof(Message), &msg1)` となる。

型 i_1, \dots, i_n の n 個のインタフェースパラメータを持つパラメータ付コマンド (あるいはイベント) はインタフェースパラメータ式 e_1, \dots, e_n をつけて呼ばれる。以下のように: `call [e1, ..., en](...)` (あるいは `signal [e1, ..., en](...)`)。インタフェースパラメータ式の e_i は型 i_i に割付けられなければならない。実際のインタフェースパラメータ値は型 i_i にキャストされた e_i である。例えば `SendMsg` 型のインタフェース `Send[uint8_t id]` を使うモジュールにおいては、

```

int x = ...;
call Send.send[x + 1](1, sizeof(Message), &msg1);

```

コマンドやイベントの実行は直ちに行われる。つまり、`call` や `signal` は関数呼び出しと同じ動作をする。`call` や `signal` 式によって実行される、実際のコマンドやイベントの実装はプログラムのコンフィグレーションでのワイアリング文によって決められている。ワイアリング文は 0、1、あるいはそれ以上の実装が実行されるよう指定することができる。1つ以上の実装が実行される場合、そのモジュールのコマンドやイベントは“fan-out”(訳注: 扇が広がる様子) すると言う。

モジュールは呼び出したリシグナルする利用コマンドやイベント のデフォルトの実装を規定することができる。提供コマンドやイベントに対してデフォルトの実装がある場合はコンパイル時にエラーとなる。デフォルトの実装は がどのコマンドやイベントの実装にもつなげていない場合に実行される。デフォルトコマンドやイベントはその実装の前に `default` という予約語をつけて定義される。

declaration-specifiers: also

```

default declaration-specifiers

```

例えば、`SendMsg` 型の `Send` インタフェースを利用するモジュールにおいて:

```

default command result_t Send.send(uint16_t address, uint8_t length,
                                   TOS_MsgPtr msg) {
    return SUCCESS;
}
/* call is allowed even if interface Send is not connected */
... call Send.send(1, sizeof(Message), &msg1) ...

```

Section7.4 では call や signal 式によってどのコマンドやイベントの実装が実際に実行されるか、どんな結果が返されるかについて述べる。

6.3 タスク

タスクとは記憶クラス task で戻り値も引数も持たない関数によって定義される、独立した制御単位である: task void myTask() { ... }³ タスクは前方宣言することもできる: task void myTask();

タスクはタスク関数の呼び出しの前に post をつけてポストされる: post myTask(); この post 実行は直ちに完了する。戻り値が 1 の場合はタスクは独立した実行をすべくポストされた場合であり、そうでない場合は 0 が返る。post 式の型は unsigned char である。

storage-class-specifier: also one of
task

call-kind: also one of
post

nesC の並列処理モデルについてはタスクを含めて、Section8 で詳しく述べる。

6.4 Atomic 文

atomic-stmt:
atomic *statement*

上記 Atomic 文はあたかも他の処理が同時には走らない(割り込まれない)かのように文(statement)が実行されることを保証するものである。並行処理でのデータ構造の変更などの排他制御に使われる。簡単な例を以下に示す:

```

bool busy; // global

void f() {
    bool available;

    atomic {
        available = !busy;
        busy = TRUE;
    }
    if (available) do_something;
    atomic busy = FALSE;
}

```

Atomic セクションは短くするべきである。この要求を満たすために nesC では atomic 文中でのコ

³ nesC関数で引数をもたないものは(void)ではなく()で宣言する。Section10.1 参照。

マンドの呼び出しやイベントのシグナルを禁止している。以下の制御構造も atomic 文中では禁止されている (nesC の将来のバージョンで許されるかもしれない): goto, return, break, continue, case, default とラベル。

Section8 では atomic と nesC の並列処理モデルとデータ競合の関係について述べる。

7. コンフィグレーション

コンフィグレーションは他のコンポーネントの集合と接続したりワイアリングして、コンポーネントの規定を実装するものである。

configuration-implementation:

```
implementation { component-listopt connection-list }
```

コンポーネントリスト (*component-list*) はこのコンフィグレーションを構築するために使われるコンポーネントのリストであり、接続リスト (*connection-list*) はそれらが互いに、あるいはコンフィグレーションの規定部にどのようにワイアリングされるかを規定する。

以後、コンフィグレーションの規定部内の規定要素を外部規定要素と呼び、コンフィグレーションのコンポーネント内の規定要素を内部規定要素と呼ぶ。

7.1 内包コンポーネント

component-list はそのコンフィグレーションを構築するために使われるコンポーネントを規定する。これらのコンポーネントは、コンフィグレーションの規定要素との名前の衝突をさけるためあるいはコンフィグレーションが利用するコンポーネントの差し替えを簡単にするため(ワイアリング自体を変更しなくてよいように)、コンフィグレーション内で別の名前をつけることができる。コンポーネントに対してつけられた名前はコンポーネントの実装部スコープに属する。

component-list:

```
components  
component-list components
```

components:

```
components component-line ;
```

component-line:

```
renamed-identifier  
component-line , renamed-identifier
```

renamed-identifier:

```
identifier  
identifier as identifier
```

二つのコンポーネントが *as* を使って同じ名前をつけられた場合はコンパイル時にエラーとなる(例: *components X, Y as X*)。

コンポーネントはただ一つのインスタンスしか持たない。コンポーネント *K* が2つの別のコンフィグレーションで利用されても(あるいは同じコンフィグレーションで2回使われた場合でさえ)、プログラム内には *K* のインスタンス(とその変数)が一つしか存在しない。

7.2 ワイヤリング(Wiring)

ワイアリングは規定要素(インタフェースやコマンド、イベント)を互いに接続するために使う。このセクションと次のセクション(7.3)では構文とコンパイル時の規則を述べる。Section 7.4では、プログラムのワイアリング文(wiring statements)が *call* や *signal* 式においてどの関数が呼び出されるかを、どうやって決定するかについて詳しく述べる。

connection-list:
connection
connection-list connection

connection:
endpoint = endpoint
endpoint -> endpoint
endpoint <- endpoint

endpoint:
identifier-path
identifier-path [argument-expression-list]

identifier-path:
identifier
identifier-path . identifier

ワイアリング文は二つの終点 (*endpoints*) をつなく。*endpoint* の *idengifier-path* は規定要素を指す。引数式リスト (*argument-expression-list*) はインタフェースパラメータ値を指定する場合に使う。*endpoint* の規定要素がパラメータ付の場合は、*endpoint* がパラメータ付である、とか *endpoint* はパラメータ値を持たない、などと言う。*endpoint* がパラメータ値を持っていて、かつ、以下のいずれかが真であればコンパイル時にエラーとなる。

- ・ パラメータ値が定数式でない
- ・ *endpoint* の規定要素がパラメータ付でない
- ・ 規定要素でのパラメータよりも多い(あるいは少ない)パラメータ値がある
- ・ パラメータ値が規定要素のパラメータ型の範囲外

endpoint の *identifier-path* が以下の 3 つの形式のいずれにもあてはまらない場合はコンパイル時にエラーとなる。

- ・ *X* *X* は外部規定要素名
- ・ *K.X* *K* は *component-list* 内のコンポーネントでかつ *X* は *K* の規定要素
- ・ *K* *K* は *component-list* 内の 1 つのコンポーネントの名前である。この形式は Section 7.3 で説明する、暗黙の接続で使われる。また、この形式はパラメータ値が指定されている場合には使えないことに注意すること。

nesC には以下の 3 つのワイアリング文がある。

- ・ *endpoint₁ = endpoint₂* (同等ワイアリング): 外部規定要素を必ず含む接続。2 つの規定要素を等しくする効果を持つ。

S₁ を *endpoint₁* の規定要素とし、*S₂* を *endpoint₂* の規定要素とする。以下の 2 つの条件のうち 1 つが満たされなければコンパイル時にエラーとなる。

- ・ *S₁* は内部規定要素で *S₂* は外部規定要素 (あるいは逆も同様) であり、*S₁* と *S₂* はどちらも提供規定要素か、どちらも利用規定要素である。
- ・ *S₁* と *S₂* はいずれも外部規定要素であり片方が提供規定要素で一方は利用規定要素である。

- ・ $endpoint_1 \rightarrow endpoint_2$ (リンクワイアリング): 2つの内部規定要素を含む接続である。リンクワイアリングは $endpoint_1$ に規定されている利用規定要素を $endpoint_2$ に規定されている提供規定要素につなぐ。これらの2つの条件が満たされなければコンパイル時にエラーとなる。
- ・ $endpoint_1 \leftarrow endpoint_2$ は $endpoint_2 \rightarrow endpoint_1$ と同じである。

3種類のワイアリングにおいて2つの規定要素は互換でなければならない。すなわちどちらもコマンドであるかイベントであるか、あるいはインタフェースインスタンスである。さらにそれらがコマンド(イベント)ならばどちらも同じ関数名でなければならないし、インタフェースであるならば同じインタフェース型でなければならない。これらの条件が満たされなければコンパイルエラーとなる。

片方の endpoint がパラメータ付であれば、もう一方もパラメータ付でかつ同じパラメータ型を持たなければならない。そうでなければコンパイルエラーとなる。

同じ規定要素は何度も接続できる。

```
configuration C {
  provides interface X;
} implementation {
  components C1, C2;

  X = C1.X;
  X = C2.X;
}
```

この例では複数のワイアリングによってインタフェース X のイベントに対しては複数のシグナル("fan-in")が、インタフェース X のコマンド呼び出し時には複数の関数が実行される("fan-out")。複数のワイアリングは2つのコンフィグレーションが別々に同じインタフェースをワイアリングした場合にも起こる。例:

```
configuration C { }
implementation {
  components C1, C2;

  C1.Y -> C2.Y;
}

configuration D { }
implementation {
  components C3, C2;

  C3.Y -> C2.Y;
}
```

全ての外部規定要素はワイアリングされなければコンパイルエラーとなる。しかし内部規定要素はワイアリングされないままでもよい(これらは他のコンフィグレーションでワイアリングされるかもしれないし、このモジュールが適切なデフォルトのイベントやコマンド実装をしていればワイアリングされないままかもしれない)。

7.3 暗黙の接続

$K_1 \leftarrow K_2.X$ や $K_1.X \leftarrow K_2$ といった書き方ができる(=や->も同じ)。この構文は $K_1.Y \leftarrow K_2.X$ が正しい接続となるような規定要素 Y をみつけるまで K_1 の規定要素を通して繰り返される($K_1.X \leftarrow K_2$ の場合は、 $K_1.X \leftarrow K_2.Y$ となるように K_2 の規定要素に対して繰り返す)。適切な Y が見つければ接続が成立するが、そうでなければコンパイルエラーとなる。

例:

```
module M1 {
  provides interface StdControl;
  ...
}
configuration C { }
implementation {
  components M1, M2;
  M2.SC -> M1;
}
}

module M2 {
  uses interface StdControl as SC;
  ...
}
```

M2.SC -> M1 は M2.SC -> M1.StdControl と同義である。

```
interface X {
  command int f();
  event void g(int x);
}

configuration C {
  provides interface X;
  provides command void h2();
}

implementation {
  components M;
  X = M.P;
  M.U -> M.P;
  h2 = M.h;
}
```

図1:簡単なワイアリングの例

7.4 ワイアリングの意味

まずパラメータ付のインタフェースを除いてワイアリングの意味を説明する。Section7.4.1でパラメータ付のインタフェースをカバーし、最後に Section7.4.2 でアプリケーション全体としてのワイアリング文への要求仕様を述べる。

中間関数⁴の観点からワイアリングの意味を定義する。あらゆるコンポーネント毎のすべてのコマンドやイベント 毎に一つの間関数 I が存在する。例えば、図 1 のモジュール M は中間関数 $I_{M.P.f}$, $I_{M.P.g}$, $I_{M.U.f}$, $I_{M.U.g}$, $I_{M.h}$ を持つ。例においてはコンポーネントやインタフェースインスタンス命や関数名に基づいて中間関数の名前を決めることとする。

中間関数は利用されるかあるいは提供される。それぞれの間関数は対応するコンポーネントの規定部にあるコマンドやイベントと同じ引数を持つ。中間関数 I の実体は他の中間関数の呼び出しのリスト(連続して実行される)である。この他の中間関数はアプリケーションのワイアリング文によって I に接続された関数である。 I が受け取る引数は、 I が呼び出す中間関数へと変更されずに渡される。 I の結果は結果のリストであり(リストの要素の型は I に対応したコマンドやイベントの戻り値の型である)、 I が呼び出した中間関数の戻り値が結合されたものである。空の結果リストを返す中間関数は接続されていないコマンドやイベントということである。2 つ以上の要素を返す中間関数は”fan-out”ということである。

⁴ nesCは中間関数を明確に分けずにコンパイルできる、そのためこのsectionで述べる動作はパラメータ付のコマンドやイベントに必要なディスパッチや実際の呼び出し以上に実行時のコストがない。

中間関数とコンフィグレーション

コンフィグレーション中のワイアリング文は中間ファイルの実体を規定する。規定要素よりは中間関数を参照するためにワイアリング文は展開される。次に = と -> の違いを除去する。中間関数 I_1 と I_2 間の接続は $I_1 <-> I_2$ と記述される。例えば図 1 のコンフィグレーション C は以下の中間ファイルの接続を規定する。

$$\begin{aligned} I_{C.X.f} <-> I_{M.P.f} & \quad I_{M.U.f} <-> I_{M.P.f} & \quad I_{C.h2} <-> I_{M.h} \\ I_{C.X.g} <-> I_{M.P.g} & \quad I_{M.U.g} <-> I_{M.P.g} \end{aligned}$$

コンフィグレーション C の $I_1 <-> I_2$ 接続では、2 つの中間関数のうちの 1 つが callee (呼び出される側) であり、もう 1 つが caller (呼び出し側) である。この接続は、caller の実体に callee への呼び出しを加えることを単純に指定するものである。以下の条件のいずれかが満たされていれば I_1 (同様に I_2) は callee である (規定要素の内部・外部という用語は、この接続を含むコンフィグレーション C に関して使っている)。

- ・ I_1 が提供コマンドかイベントの内部規定要素に相当する
- ・ I_1 が利用コマンドかイベントの外部規定要素に相当する
- ・ I_1 がインタフェースインスタンス X のコマンドに相当し、かつ X は内部提供、あるいは外部利用規定要素である
- ・ I_1 がインタフェースインスタンス X のイベントに相当し、かつ X は外部提供、あるいは内部利用規定要素である

これらのどの条件も満たされないとき、 I_1 は caller となる。Section 7.2 でのワイアリング規則は接続 $I_1 <-> I_2$ では caller 同士や callee 同士を結ぶことができないことを保証する。図 1 のコンフィグレーション C において、 $I_{C.X.f}$ 、 $I_{C.h2}$ 、 $I_{M.P.g}$ 、 $I_{M.U.f}$ は caller であり、 $I_{C.X.g}$ 、 $I_{M.P.f}$ 、 $I_{M.U.g}$ 、 $I_{M.h}$ は callee である。したがって C の接続は $I_{M.P.f}$ の呼び出しが $I_{C.X.f}$ に、 $I_{C.X.g}$ の呼び出しが $I_{M.P.g}$ へ追加されることを示している。

中間関数とモジュール

モジュール内の C コードは中間関数を呼び出したり、あるいは中間関数から呼び出されたりする。

モジュール M の提供コマンド・イベント に対する中間関数 I は M 内の の実装の単一呼び出しを含む。その結果はこの呼び出し結果の 1 つのリストである。

call (e_1, \dots, e_n) という式は以下のように評価される:

- ・ 引数の e_1, \dots, e_n が評価され、値 v_1, \dots, v_n が得られる
- ・ に対応する中間関数 I が引数 v_1, \dots, v_n で呼び出され、リスト L を結果として返す
- ・ L が (w) であれば (単一要素リスト)、call の戻り値は w である。
- ・ L が (w_1, w_2, \dots, w_m) であれば (2 以上の要素を持つリスト)、call の戻り値は の戻り値型 による。 が void であれば、結果は void である。それ以外では は結合関数 c (Section 10.3 でどのようにして結合関数が型に結び付けられるかを示す) に結び付けられなければならない、そうでなければコンパイルエラーとなる。結合関数は型 の 2 つの引数を取り、型 の戻り値を返す。call の戻り値は $c(w_1, c(w_2, \dots, c(w_{m-1}, w_m)))$ である (L の要素の順序は任意だったことに注意すること)。
- ・ L が空の場合、 のデフォルト実装が引数 v_1, \dots, v_n で呼び出され、その戻り値が call の戻り値となる。Section 7.4.2 では L が空でかつ のデフォルト実装がない場合にコンパイルエラーとなることを示す。

signal 式の規則も同じである。

中間関数の例

図2は図1のコンポーネントに対して生成された中間関数を示したものである。図中、次のようなCライクな構文が使われている: `list(x)`は `x` を要素に持つ単一要素のリストを生成するものであり、`empty_list` は要素が空のリストの定数、`list_concat` (訳注: `concat_list` は誤り)は2つのリストを結合する。`M.P.f`、`M.U.g`、`M.h`の呼び出しはモジュールMのコマンドやイベントの実装(ここには示さない)の呼び出しを示す。

<pre>list of int IM.P.f() { return list(M.P.f()); }</pre>	<pre>list of void IM.P.g(int x) { list of int r1 = IC.X.g(x); list of int r2 = IM.U.g(x); return list_concat(r1, r2); }</pre>
<pre>list of int IM.U.f() { return IM.P.f(); }</pre>	<pre>list of void IM.U.g(int x) { return list(M.U.g(x)); }</pre>
<pre>list of int IC.X.f() { return IM.P.f(); }</pre>	<pre>list of void IC.X.g(int x) { return empty_list; }</pre>
<pre>list of void IC.h2() { return IM.h(); }</pre>	<pre>list of void IM.h() { return list(M.h()); }</pre>

図2: 図1の中間関数

7.4.1 ワイヤリングとパラメータ付関数

コンポーネント K のコマンド・イベント c が型 t_1, \dots, t_n のインタフェースパラメータを持つ場合、それぞれ個別のパラメータの組 $(v_1: t_1, \dots, v_n: t_n)$ に対応する中間関数 I_{v_1, \dots, v_n} が存在する。

モジュールにおいて、中間関数 I_{v_1, \dots, v_n} がパラメータ付の提供コマンド(あるいはイベント) c に対応する場合、 I_{v_1, \dots, v_n} 内での c の実装への呼び出しでは c のインタフェースパラメータとして v_1, \dots, v_n が渡される。

式 `call [e1, ..., em](e1, ..., en)` は以下のように評価される:

- ・ 引数 e_1, \dots, e_n が評価され、値 v_1, \dots, v_n が得られる
- ・ 引数 e'_1, \dots, e'_m が評価され、値 v'_1, \dots, v'_m が得られる
- ・ v_i は型 t_i にキャストされる。 t_i は、 c の i 番目のインタフェースパラメータの型である
- ・ c に対応する中間関数 I_{v_1, \dots, v_n} は引数 v_1, \dots, v_n で呼び出され、リスト L を結果として返す⁵
- ・ L が1つ以上の要素を持っていれば、`call` の戻り値はパラメータなしの場合と同様に作られる
- ・ L が空の場合は c のデフォルト実装が、インタフェースパラメータ値 v'_1, \dots, v'_m と引数 v_1, \dots, v_n で呼び出され、その戻り値が `call` の戻り値となる。Section 7.4.2 では L が空でかつ c のデフォルト実装がない場合にコンパイルエラーとなることを示す。

signal 式の規則も同じである。

⁵ この呼び出しはたいてい、実行時に呼ぶべきものをいくつかのコマンドの実装から選ぶということを含む。中間関数にかかる実行時のコストはこれだけである。

ワイアリング文中の endpoint がパラメータ付の規定要素に帰するのは以下の2つの場合である:

- ・ endpoint がパラメータ値 v_1, \dots, v_n を指定している。endpoint がコマンド・イベント $1, \dots, m$ に対応する場合、対応する中間関数は $I_{1,v_1, \dots, v_n}, \dots, I_{m,v_1, \dots, v_n}$ であり、結合は前述のとおりのものである。
- ・ endpoint がパラメータ値を指定しない。この場合はワイアリング文中の両方の endpoint は、インタフェースパラメータ型 $1, \dots, n$ と同じパラメータ付の規定要素に対応する。片方の endpoint がコマンド・イベント $1, \dots, m$ に対応しもう一方が $1, \dots, m$ に対応するなら、接続 $I_{i,w_1, \dots, w_n} \leftarrow I_{i,w_1, \dots, w_n}$ がすべての $1 \leq i \leq m$ を満たす i 個別のパラメータの組 ($w_1: 1, \dots, w_n: n$) に対して存在する (すなわち endpoint はすべての対応するパラメータ値に対して接続される)。

7.4.2 アプリケーションレベルの要求

アプリケーションのワイアリング文が必ず満たさなければならない、以下の2つの要求がある。これが満たされなければコンパイルエラーとなる:

- ・ 中間関数だけを含む無限ループが存在してはいけない
- ・ アプリケーションモジュールでの `call` (あるいは `signal`) 式のそれぞれについて、
 - ・ 呼び出しがパラメータ無しの場合: 呼び出しが空の結果リストを返す場合は のデフォルト実装がなければならない (結果リストの要素数はワイアリングによってのみ決定される)
 - ・ 呼び出しがパラメータ付の場合: のインタフェースパラメータの値の代入が空の結果リストを返す場合は、 のデフォルト実装がなければならない (与えられたパラメータ値の組に対する結果リストの要素数はワイアリングによってのみ決定される)この条件は、呼び出し位置でのインタフェースパラメータ値を指定するための式についてはあてはまらない。

8 nesC の並列処理

nesC では、処理完了まで走りきる (訳注: 以後、場所によっては run-to-completion のまま用いる) タスク (通常、実行中の処理を表す) とハードウェアから非同期にシグナルされる割り込みハンドラとからなる実行モデルを想定している。nesC のスケジューラはタスクをどんな順番でも実行できるが、run-to-completion の規則には従わなければならない (TinyOS の標準スケジューラは FIFO ポリシーに従う)。タスクは処理を中断 (preempt) されず処理完了まで走りきるため、タスク同士に関しては atomic (分離不可能) であるが、割り込みハンドラに対しては atomic ではない。

このような並列処理モデルであるため、nesC プログラムは競合が起こる状況、特にプログラムの共有物、つまりグローバル変数やモジュール変数 (nesC には動的なメモリ確保の機能はない)、のデータ競合に敏感である。タスクからのみ共有物にアクセスするか、atomic 文内でのみアクセスすることで競合は避けられる。nesC コンパイラはコンパイル時に潜在的なデータ競合をプログラマに教えてくれる。

形式的に、nesC プログラムのコードを 2 つの種類に分ける:

同期コード (Synchronous Code=SC): タスクからのみ到達可能なコード (関数、コマンド、イベント、タスク)

非同期コード (Asynchronous Code=AC): 少なくとも 1 つの割り込みハンドラから到達可能なコード

non-preemption (訳注: タスクが互いに割り込みあわないことを指す) によりタスク間ではデータ競合は起こらないが、SC と AC、同様に AC と AC との間では依然潜在的な競合が存在する。一般的に AC から到達可能な共有物の更新はすべて潜在的なデータ競合となる。nesC が約束する基本的な不変条件は次のとおりである:

無競合の不変条件: 共有物のすべての更新は SC のみか atomic 文内かいずれかで行う。f のすべての呼び出しが atomic 文中にある限りは、atomic 文から呼び出される関数 f の実体は atomic 文の中にあるとみなされる。

コンパイラが検出できない競合条件を作り出すことは可能であるが、それは複数の atomic 文やタスクにわたり、媒介する変数で storage を使う場合である。

nesC は実際には起こりえないデータ競合を報告することもある。例えばあるほかの変数のガードによってすべてのアクセスが保護されている場合である。このような場合冗長な (コンパイル) メッセージを避けるため、プログラマは変数 v に関するすべてのデータ競合 warning を出ないようにするために、norace 記憶クラス指定子を変数 v につけて注釈することができる。ただし norace は注意して使うべきである。

nesC は AC であるのに async 付で宣言されていないコマンドかイベントがあるとコンパイルエラーを報告する。これは割り込みハンドラ内で安全に実行できるようになっていないコードがうっかり呼び出されないようにする。

9 nesC アプリケーション

nesC アプリケーションは 3 つの部分からなる: C 宣言と定義のリスト、インタフェースの組とコンポーネントの組である。nesC アプリケーションの命名環境は以下のように形作られている:

- ・ 最も外側の、3 つの名前空間を持つグローバルスコープ: C 宣言と定義のための C 変数と C タグ名前空間、nesC インタフェース型とコンポーネントのためのインタフェース型とコンポーネント名前空間。
- ・ 通常の (訳注: C) のように、C 宣言と定義はグローバルスコープ内にそれ自身の入れ子になったスコープをもたらすことができる (関数宣言・定義、関数内のコードブロックなど)。
- ・ それぞれのインタフェース型はインタフェースコマンドやイベントを持つスコープをもたらす。このスコープはグローバルスコープから入れ子になり、そのためコマンドやイベントの定義はグローバルスコープ内に定義された C の型やタグを参照できる。
- ・ それぞれのコンポーネントは 2 つの新しいスコープをもたらす。1 つは規定部スコープでグローバルスコープの入れ子で、コンポーネントの規定要素を持つ、変数のための名前空間を含む。もう 1 つが実装部スコープで規定部スコープの入れ子で、変数とタグのための名前空間を含む。

コンフィグレーションに関しては、実装部スコープの変数名前空間が、コンポーネント内に含むコンポーネントを参照するために使う名前を含む (Section 7.1 参照)。モジュールに関しては、実装部スコープが、モジュール本体内のタスクや C 宣言と定義を含む。これらの宣言などは実装部スコープ内にさらに入れ子された自分自身のスコープを持つこともある (関数本体やコードブロックなど)。このスコープの入れ子構造により、モジュール内のコードはグローバルスコープにある C 宣言・定義へアクセスできるが、他のコンポーネントないの宣言や定義にはアクセスできない。

nesC アプリケーションでの C 宣言や定義、インタフェース型やコンポーネントは on-demand ローディングプロセスによって決定される。nesC コンパイラへの入力は 1 つのコンポーネント K である。nesC コンパイラは最初に C ファイル `tos` をロードし (Section 9.1)、次にコンポーネント K をロードする (Section 9.2)。アプリケーションのコードはこれら 2 つのファイルをロードする一連のプロセスの一部としてロードされるコードのすべてである。nesC コンパイラは、spontaneous 属性 (Section 10.3) のないすべての関数、コマンド、イベント呼び出しはロードしたコード内で行われるものとする (すなわち、spontaneous でない関数に対する「見えない」呼び出しはない)⁶。

ロードしたファイルのプリプロセス処理において、nesC は NESC シンボルを nesC 言語とコンパイラのバージョンを識別する番号 XYZ として定義する。nesC 1.1 では XYZ は少なくとも 110 である⁷。

ロードした C ファイル、nesC コンポーネントあるいはインタフェースの処理の中には、対応するソースファイルの配置することも含まれる。ファイルを配置するのに使われる仕組みは本リファレンスマニュアルの範囲外である。現在のコンパイラがどのようにしてこれを行うかは、ncc の man ページを参照してほしい。

9.1 C ファイル X のロード

X がすでにロードされていれば、それ以上何も行われない。そうでなければファイル `X.h` が配置されプリプロセスが行われる。(#define や #undef によって行われる) C マクロへの変更は、続けてプリプロセスされる全てのファイルからみえる。プリプロセスをした `X.h` ファイル内の C 宣言や定義は C グローバルスコープに置かれ、そのため続けてプリプロセスされる C ファイル、インタフェース型、コンポーネントからみえる。

⁶ 例えば、現バージョンの nesC コンパイラはどこからも呼び出されないコードを除去するためにこの情報を使う。

⁷ NESC シンボルは 1.1 より前のバージョンの nesC では定義されなかった。

9.2 コンポーネント K のロード

K がすでにロードされていれば、それ以上何も行われぬ。そうでなければファイル $X.nc$ が配置されプリプロセスが行われる。(#define や #undef によって行われる) C マクロへの変更は取り除かれる。プリプロセスされたファイルは以下の文法を用いて解釈される:

```
nesC-file:
    includes-listopt interface
    includes-listopt module
    includes-listopt configuration
```

```
includes-list:
    includes
    includes-list includes
```

```
includes:
    includes identifier-list ;
```

$X.nc$ が module K か configuration K を定義していなければコンパイルエラーとなる。そうでなければ、includes-list に指定されているすべての C ファイルがロードされる (Section 9.1)。続けてコンポーネントの規定部で使われる全てのインタフェース型がロードされる (Section 9.3)。次に、コンポーネントの規定部が処理される (Section 5)。 K がコンフィグレーションならば K によって指定されたすべてのコンポーネント (Section 7.1) がロードされる (Section 9.2)。最後に K の実装部が処理される (Section 6, 7)。

9.3 インタフェース型 I のロード

I がすでにロードされていれば、それ以上何も行われぬ。そうでなければファイル $X.nc$ が配置されプリプロセスが行われる。(#define や #undef によって行われる) C マクロへの変更は取り除かれる。プリプロセスされたファイルは先に述べた nesC-file の規則にしたがって解釈される。 $X.nc$ が interface I を定義していなければコンパイルエラーとなる。そうでない場合は includes-list に指定されているすべての C ファイルがロードされる (Section 9.1)。そして I の定義が処理される (Section 4)。

以下はコンポーネントやインタフェース内で C ファイルをインクルードする例である。インタフェース型 Bar は Bar で使う型を定義している C ファイル $BarTypes.h$ を include しようとしている。

<pre>Bar.nc: includes BarTypes; interface Bar { command result_t bar(BarType arg1); }</pre>	<pre>BarTypes.h: typedef struct { int x; double y; } BarType;</pre>
---	---

インタフェース Bar の定義は $BarType$ を参照できるし、インタフェース Bar を利用あるいは提供するどんなコンポーネントも同じように ($BarType$ を) 参照できる (インタフェース Bar 、とそれから引かれる $BarTypes.h$ はすべてのコンポーネント規定部や実装部が処理される前にロードされる)。

10 その他

10.1 引数のない関数、古いスタイルの C 宣言

nesC で引数のない関数は () で宣言されるのであり、(void) ではない。後者の構文はコンパイルエラーとなる。

古いスタイルの C 宣言 (() の宣言) や関数定義 (引数リストの後ろで指定されるパラメータ) はインタフェースやコンポーネント内では許されない (コンパイルエラーになる) 。

これらは C ファイルには何の影響もない (したがって既存の .h ファイルは何も変更しなくてよい) 。

10.2 コメント

nesC は // のコメントを C、インタフェース型、コンポーネントファイル内で許す。

10.3 属性 (Attributes)

nesC は gcc⁸ の `__attribute__` 構文をプロパティや関数、変数、typedef に使う。これらの属性は宣言部 (宣言子の後ろ) か関数定義 (パラメータリストの後ろ)⁹ のいずれかに置かれる。x の属性は x のすべての宣言と定義にかかるすべての属性をあわせたものである。

nesC の属性の構文は以下のとおりである：

```
init-declarator-list: also  
                  init-declarator attributes  
                  init-declarator-list , init-declarator attributes
```

```
function-definition: also  
                  declaration-specifiersopt declarator attributes  
                  declaration-listopt compound-statement
```

```
attributes:  
              attribute  
              attributes attribute
```

```
attribute:  
              __attribute__ ( ( attribute-list ) )
```

```
attribute-list:  
              single-attribute  
              attribute-list , single-attribute
```

```
single-attribute:  
              identifier  
              identifier ( argument-expression-list )
```

nesC は以下の 3 つの属性をサポートする。

- ・ `C`: この属性はモジュールのトップレベルでの C 宣言や定義 *d* に使われる (他の宣言に対して無視される)。これは *d* がモジュール内のコンポーネント毎の実装部スコープではなく、グローバルの C スコープに現れるように指示する。これにより *d* は C コードから使うことができる (*d* が関数なら呼ぶことができる)。

⁸ <http://gcc.gnu.org>

⁹ gcc は関数定義内の引数リストの後ろの属性を許さない

- ・ `spontaneous`: この属性はどんな関数 f (モジュール内でも C コード内でも) にも使うことができる。これはソースコード内に現れない f の呼び出しがあることを示す。典型的には自発的に (`spontaneously`) 呼ばれる関数は割り込みハンドラと C の `main` 関数である。Section 9 では nesC コンパイラが、コンパイル時にどのようにして `spontaneous` 属性を使うかについて述べた。
- ・ `combine(fnname)`: この属性は `typedef` 宣言中の型に対して結合関数を指定する。結合関数は "fan-out" するコマンドやイベントの複数の戻り値をどうやって束ねるかを示す。例:

```
typedef uint8_t result_t __attribute__((combine(rcombine)));

result_t rcombine(result_t r1, result_t r2)
{
    return r1 == FAIL ? FAIL : r2;
}
```

この例では戻り値の型が `result_t` であるコマンド(あるいはイベント)を束ねる処理は論理的なように規定している。

型 t に対する結合関数 c が次の型を持たなければコンパイルエラーとなる。 $t\ c(t, t)$

属性の利用例: `RealMain.td`:

```
module RealMain { ... }
implementation {
    int main(int argc, char **argv) __attribute__((C, spontaneous)) {
        ...
    }
}
```

この例では関数 `main` が、リンカが `main` を見つけられるように、C のグローバルスコープに実際に現れるように宣言するものである。これまた、`main` がプログラム中のどこにも `main` の呼び出しがなくても `main` が呼ばれることも宣言している (`spontaneous`)。

10.4 コンパイル時定数関数

nesC には新しい種類の定数式がある: 定数関数という。これらはコンパイル時に評価されて定数になる、言語内に定義された関数である。

現在の nesC では以下の 2 つの定数関数がある:

- ・ `unsigned int unique(char *identifier)`
戻り値: 同じ `identifier` 文字列での `unique` 呼び出しが n 回ある場合、それぞれの呼び出しで異なる、 $0 \dots n-1$ の範囲の符号無し整数が返される。
`unique` は、ユニークな整数値をパラメータ付のインタフェースインスタンスに渡すために使うよう意図されている。これによってパラメータ付のインタフェースを提供するコンポーネントがそのインタフェースにつながったそれぞれのコンポーネントを識別することができる。
- ・ `unsigned int uniqueCount(char *identifier)`
戻り値: 同じ `identifier` 文字列での `unique` 呼び出しが n 回ある場合、`uniqueCount` は n を返す。
`uniqueCount` は、`unique` が返す値をインデックスとして使う多次元配列(あるいは他のデータ構造)のために使うよう意図されている。例えば、`Timer` サービスはパラメータ付インタフェースと `unique` を使ってサービスのクライアント(とそれぞれのタイマー)を識別しているが、タイマーデータの構造体を正しい数だけ割り当てるために `uniqueCount` を使う。

(以下、Appendix A 以降は原文を参照してください)

GNU Free Documentation License
Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal,

commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this

License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in

the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such

parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.