T-Engine Forum
Specification

TEF040-S103-1.01.01/ja
2006-06-06

T-Format (3): C 言語グローバルシンボル名

T-Format (3): Global Symbol Naming Rule in C Language

Number:	TEF040-S103-1.01.01/ja
Title:	T-Format (3): C 言語グローバルシンボル名
	T-Format (3): Global Symbol Naming Rule in C Language
Status:	[ ] Working Draft, [ ] Final Draft for Voting, [X] Standard
Date:	2003/02/28 First Edited
	2003/08/01 Updated to 1.01.00
	2003/10/27 Updated to 1.01.01 ライブラリファイル命名規則他を追記
	2006/06/06 Updated to 1.01.02 一部誤植訂正

目 次

はじめに (Foreword)

あらゆるものにコンピュータが入りネットワークでつながれるユビキタス・コンピューティング環境の構築を目指した、オープンなリアルタイムシステム標準開発環境を提供するため、T-Engine プロジェクトが発足した。 T-Engine は携帯情報機器やネットワーク接続型の家電機器などを効率良く短期間で開発するのに最適な開発環境の提供を目指している。T-Engine は eTRON と呼ばれるプロジェクトのネットワークセキュリティアーキテクチャに対応し、安全でない通信路を介した通信においても、盗聴、改竄、なりすましを防御して安全に目的の相手に電子情報を送る機構を備える。

効率のよい開発をサポートするために、規格化されたハードウェア（T-Engine ボード）、標準リアルタイムカーネル（T-Kernel）を定め、ミドルウェアを流通させることに特に力を入れている。 また、T-Engine は半導体メーカー、ハードウェアメーカー、ソフトウェアメーカー、システムメーカーの連携を円滑にし、相互のビジネスを活発化し、開発期間や開発コストの低減により付加価値の高い製品を短期間で提供することを狙っている。更に、T-Engine は高度な半導体技術や実装技術、ソフトウェア技術を採用し、 他に追随を許さない先進的な応用製品の開発を行う。

規定範囲 (Scope)

T-Format は T-Engine、T-Kernel 上で動作するミドルウェアやアプリケーションソフトウェアのコード形式を規定する。T-Format には、以下の 3 種類の規定が含まれる。

1. ソースコードスタイルガイドライン
   T-Engine、T-Kernel 上で動作するミドルウェアやアプリケーションソフトウェアのソースコードのスタイル形式。異なるベンダーが作成したソースコードを組み合わせてコンパイル、リンクできるための規定である。

2. 標準バイナリ形式
   T-Engine、T-Kernel 上で動作するミドルウェアやアプリケーションソフトウェアがバイナリコードで配布される際の標準実行形式。実行コード形式とデバッガシンボル形式を規定する。

3. 標準ドキュメント形式
   流通するミドルウェアやアプリケーションソフトウェアに添付するドキュメントの種類と形式に関する規定。

本仕様は上記の T-Format の一部分を構成し、C 言語によるソフトウェアのグローバルシンボルの命名規則を定めるものである。

参照規定 (Normative References)

[1] T-Engine Forum. "T-Format (2) : T-Engine Vendor Code System", TEF-040-S102, 2002.

用語定義 (Terms and Definitions)

ミドルウェア (Middleware) ミドルウェア（Middleware）は、T-Engine/T-Kernel 環境上で動作するアプリケーションソフトウェア、または他のミドルウェアに対して何らかのサービスを提供するソフトウェアを指す。

注： T-Kernel 仕様で使われている「ミドルウェア」という用語は、「サブシステム」を使って実現されたソフトウェアモジュールとして使われている（「狭義のミドルウェア」と呼ぶ）。本仕様書で用いられている用語は、これよりも広く、アプリケーションソフトウェア全般に対してサービスを提供するソフトウェアは、すべて「ミドルウェア」、または「広義のミドルウェア」と呼ぶ。

1. T-Engine ミドルウェアモデル

ここでは、T-Engine アーキテクチャ上におけるミドルウェアのアーキテクチャモデルとそれに基づく分類について述べる。

1.1 実現方式による分類

ミドルウェアは、その実現方式によって、以下の 3 種類がある。

(1) サブシステム型ミドルウェア
T-Kernel が提供するサブシステムの機構を使って実現されるミドルウェア。ユーザアプリケーションは、次の 2 種類の方法でサービスを享受する。一つ目は、サブシステムが提供する拡張 SVC を使ってアプリケーションが呼び出すことである。もう一つは各種イベントによって呼び出されるサブシステムモジュールによる提供である。

(2) デバイスドライバ型ミドルウェア
T-Kernel が提供するデバイス管理の機構を使って実現されるミドルウェア。このミドルウェアは、T-Kernel が提供する汎用的なデバイスドライバインタフェースを介して使う。

(3) ライブラリ型ミドルウェア
ユーザライブラリとして実現されるミドルウェア。ユーザアプリケーションは、該当するライブラリをコード生成時にリンクする必要がある。また、ユーザアプリケーションは、ライブラリが提供する関数を呼び出すことや、または、ライブラリが提供する帯域変数にアクセスすることによってサービスを受ける。

(4) ユーザタスク型ミドルウェア
ユーザタスクで実現されたサーバによって実現されミドルウェア。ここでは、このサーバのことを、「マネージャサーバタスク」と呼ぶ。ユーザアプリケーションは、T-Kernel が提供するタスク間通信や同期の機能を使ってサーバタスクからサービスを受ける。

1.2 提供サービスによる分類

ミドルウェアは、それが提供するサービスによって分類される。
ここでのサービスによる分類とは便宜的なもので、ミドルウェアの提供するサービスによっては、どちらにもとれることもある。

(1) デバイスドライバ型ミドルウェア
T-Engine に備えられた、または接続された単一のデバイスに対して、一定の抽象度を持った、ソフトウェアインタフェースを提供するミドルウェア。

例: RS232C ドライバ、PCMCIA ドライバ、Ethernet ドライバ、USB ドライバ、Graphic Card ドライバ

RS232C、LAN NIC、USB、ISO14443、PCMCIA といったような、通信系のデバイスは、その先に、更に別のデバイスを接続する。従って、その先に接続されるデバイスドライバの移植性を得るためには、こうした通信系デバイスのデバイスドライバの上位 API は標準化する（「第五章 標準ミドルウェア規定」参照）

また、LAN NIC や記憶メディア、プリンタなどのデバイスは、処理が複雑なため、デバイスドライバの上に更に、高抽象度のインタフェースを備えることが多く、しかもその種類も多岐にわたる。従ってその場合も、デバイスドライバの上位 API を標準化する必要がある。

その他にも、一般的な機能を持つデバイスについては、必要であれば、API の標準化をする。

(2) 抽象デバイス型ミドルウェア
T-Engine に備えられた、または接続された、単一または複数のデバイスに対して、高い抽象度を持った、ソフトウェアインタフェースを提供するミドルウェア。

例： TCP/IP、Printing I/F、FAT ファイルシステム、GUI パッケージ (ウィンドウシステム)

抽象デバイス型ミドルウェアは、基本的に、デバイスドライバ型ミドルウェアの上、または、他の抽象デバイス型ミドルウェアの上に構築されるものである。従って、前提とする下位 API は、標準化されたデバイスドライバ型ミドルウェアや他の抽象デバイス型ミドルウェアの上位 API とそろえる必要がある。また、抽象デバイス型ミドルウェアが提供する上位 API は、一般性があるか、または他のミドルウェアから呼び出される可能性がある場合は、標準化を行う。

(3) カーネル拡張型ミドルウェア
T-Kernel が管理するハードウェア資源や、ソフトウェア資源に対して、T-Kernel の機能やミドルウェアの機能を利用して、より高い抽象度のサービスを提供するためのミドルウェア。

例： プロセス管理機能、仮想記憶機能、ローダ機能、CLI(コマンドラインインタプリタ) 機能

(4) 一般機能提供型ミドルウェア
特に、カーネルが提供する資源や外部接続デバイスとは関連性が薄い、一般的な計算機能を提供するミドルウェア。

例： プロセス管理機能、仮想記憶機能、ローダ機能、CLI(コマンドラインインタプリタ) 機能

2. グローバルシンボル（Global Symbol）

本章では、T-Format のミドルウェアを相互に組み合わせて利用することを可能にすることを保証するために、「グローバルシンボル規定」を定める。主に、異なるベンダーが提供する C 言語で書かれたのミドルウェアを組み合わせて、コンパイル・リンクする際の不具合を防ぐことを目的としている。特に、ライブラリとして実現されるミドルウェアが提供するグローバルに利用可能な名前空間の識別を規定する。

2.1 T-Engine ベンダーコード（T-Engine Vendor Code）

T-Format 形式に従ったミドルウェアを提供するミドルウェアベンダーは、T-Engine ベンダーコードの割り当てを受ける [1]。

異なるベンダーが同じグローバルシンボルを利用することがないように、グローバルシンボルに、この T-Engine ベンダーコードが利用される。

（例）

ベンダ名：	YRP ユビキタスネットワーキング研究所
ドメイン名：	unl

2.2 ミドルウエアファイル名（File Name of Middleware）

T-Format のミドルウェアを構成するファイル（ライブラリ等のバイナリファイルとヘッダファイル）には、T-Engine Forum 内で唯一性が保証される命名規則を以下の通り規定する。ミドルウェアを構成するファイルを次の規則で名前をつける。(通称定義用ヘッダファイル名については 2.7節参照)

< ライブラリファイル名 > ::= “lib” < ミドルウェア名 > “.” < 拡張子 >
< その他ファイル名 > ::= < ミドルウェア名 > “.” < 拡張子 >
< ミドルウェア名 > ::= < ソフトウェアベンダ名 > “_” < 機能名 > [“_” < 詳細機能名 >]

機能名（Functionality Name）

機能名は、そのミドルウェアのファイルが実現する機能を表現する名前とする。各ベンダーの中で、名前が衝突しないように、ユニーク性を保つようにしなければならない。

機能名は、“a”～“z”、“0”～“9” の文字を使った、2 文字以上の文字列とする。

（例）

機能：	MPEG2 の CODEC
機能名：	mpeg2

詳細機能名（Detail Name）

一つのミドルウェアが、複数のバイナリファイルやヘッダファイルから構成される場合に、それらを区別するための詳細な名称。詳細機能名は、機能名同様、“a”～“z”、“0”～“9” の文字を使った、2 文字以上の文字列とする。
(例) ミドルウェアファイル名の例
上記の命名規則にのっとり、ベンダーコード「unl」を持つベンダーが作成した、MPEG2 CODEC のライブラリのミドルウェアのファイル名は以下のようになる。

```
libunl_mpeg2.a
unl_mpeg2_basic.h
```

2.3 export される関数名

あるミドルウェアのライブラリファイルに含まれており、アプリケーションに Export される関数名の命名規則を次のように定める。

＜関数名＞ ::= ＜ミドルウェア名＞ “_” ＜関数識別子＞

＜関数識別子＞ は、“A”～“Z”、“a”～“z”、“0”～“9” の文字を使った、2 文字以上の文字列で表される、その関数の機能名である。
(例)

mpeg2 エンコード関数：	`void unl_mpeg2_encode(...);`
mpeg2 デコード関数：	`void unl_mpeg2_decode(...);`

2.4 export される大域変数名

あるミドルウェアのライブラリファイルに含まれており、アプリケーションに Export される大域変数名の命名規則を次のように定める。

＜変数名＞ ::= ＜ミドルウェア名＞ “_” ＜変数識別子＞

＜変数識別子＞ は、“a”～“z”、“0”～“9” の文字を使った、2 文字以上の文字列で表される、その変数の機能名である。
（例）
mpeg2 ライブラリで使われる、etronid という変数であれば、次のように表現される。

```
char unl_mpeg2_etronid[16];
```

2.5 関数、大域変数で用いられるデータ型名

export する関数の引数や戻値で用いられるデータ構造や、大域変数で使われるデータ構造の名前の命名規則を次のように定める。

＜データ構造名＞ ::= ＜ミドルウェア名＞ “_” ＜データ構造識別子＞

＜データ構造識別子＞は、“a”～“z”、“0”～“9” の文字を使った、2 文字以上の文字列で表される、その変数の機能名である。

(例)

```
struct point {
  int x;
  int y;
};
typedef struct point unl_mpeg2_point;
```

2.6 ヘッダファイルに含まれる定数マクロ名

export する関数の引数や戻値で代入される Special Value や、大域変数に設定されうる Special Value を定義するための、定数マクロ名の命名規則を次のように定める。

＜マクロ名＞ ::= ＜ミドルウェア名（大文字）＞ “_” ＜マクロ識別子＞

(例)

```
#define UNL_MPEG2_MAXID 225
#define UNL_MPEG2_MINID 5
```

2.7 「通称」定義の推奨

上記の命名規則で生成された名前は、複数のベンダーが独立に開発したミドルウェアを一緒に利用するときに、実行コード生成時の不具合を解消するための規定である。こうした規定に則った場合、次の 2 点が問題となる。

1. 名前が長い
2. 一般に流布している名称と異なる

これらの命名規則は、他のベンダーのミドルウェアと結合するときに不具合を解消するためのもので、それだけを単体で利用するときには、不要な規定である。そこで、上記の問題を解消するために、より短くわかりやすい通称をマクロ等でつけることが推奨される。たとえば、TCP/IP のソケットインタフェースミドルウェアを、本命名規則に則ると、

```
unl_tcpip_socket();
```

といった関数になる。これを、

```
#define socket unl_tcpip_socket
```

といった通称を定義した、マクロセットを用意することが推奨される。

通称定義用ヘッダファイルの命名規則を次のように定める。

＜通称定義用ヘッダファイル名＞ ::= ＜ミドルウェア名＞ “_common.” ＜拡張子＞

(例)

unl_mpeg_common.h の内容

```
#include "unl_mpeg.h"
#define DREQ UNL_MPEG_DREQ
#define decode(x) unl_mpeg_decode(x)
```

unl_mpeg.h の内容

```
typedef struct {
...
} UNL_MPEG_DREQ;
extern int unl_mpeg_decode ( unl_mpeg_dfmt *);
```

3. ミドルウェアパッケージ名 (Middleware Package Name)

T-Engine のミドルウェアには、ミドルウェアパッケージ名を付与することができる。ミドルウェアパッケージ名は、そのミドルウェアの性質や特徴を表す名前で、ミドルウェアを扱うソフトウエア開発環境におけるパッケージの検索処理などで用いることを想定している。ミドルウェアパッケージ名は、次の規則に従うものとする。

＜パッケージ名＞ ::= ＜ミドルウェア名＞ ”.” ＜拡張子＞
＜ミドルウェア名＞ ::= ＜ソフトウェアベンダ名＞ “_” ＜機能名＞ [“_” ＜詳細機能名＞] “_” ＜バージョン表現＞

ここでいう＜機能名＞、＜詳細機能名＞は、ミドルウェアファイル名でつけたものと同じものとする。＜バージョン表現＞については、次節で定める。
(例) ミドルウェアパッケージ名の例
上記の命名規則にのっとり、ＵＮＬが作成した、MPEG2 CODEC のライブラリ型ミドルウェアのパッケージ名は以下のようになる。

unl_mpeg2_codec_010311

4. バージョン表現（Version Representation）

ミドルウェアのバージョンは、以下のとおり 5 桁の数字・文字で表現する。

xyyzz

xx, yy, zz それぞれの形式と意味を以下のように定める。

x: 0〜9: メジャーバージョン番号（major version number）

ミドルウェアにバックワードコンパチビリティがなくなる変化をもたらすバージョン変化をした時に、インクリメントする。従って、ミドルウェアのメジャーバージョン番号が変化した場合、そのミドルウェアを利用しているソフトウェアは、ソースコードの変更が必要になる場合があることを示す。

yy: a0〜z0, 00〜99: マイナーバージョン番号（minor version number）

ミドルウェアの API に変化があるが、バックワードコンパティビリティが確保されている場合は、メジャーバージョン番号はそのままで、このマイナーバージョン番号をインクリメントする。ミドルウェアのマイナーバージョンが変化した場合、そのミドルウェアを利用しているソフトウェアは、再リンクするだけで動作することを示す。

a0〜z0 はプレリリースに用いて、00〜99 は正式リリースに用いる。例えば、メジャーバージョン 1 のための α リリースは、

1.a0.zz

という番号になる。また、メジャーバージョン 3 のためのベータリリースであれば、

3.b0.zz

となる。

zz: 00〜99: マイナーリリース番号（minor release number）

ミドルウェアの API や、外部からみたときの論理的な動作にコンパティビリティを保った状態でのバージョン変化の際は、メジャーバージョン番号、マイナーバージョン番号を変化させずに、このマイナーリリース番号をインクリメントする。例えば、バグフィックスや、より効率のよいアルゴリズムの導入などがこれに相当する。ミドルウェアのマイナーバージョンが変化した場合、そのミドルウェアを利用しているソフトウェアは、再リンクするだけで動作することを示す。

なお、最初にリリースするミドルウェアのバージョン番号は、**1.00.00** とすることを基本とする。しかし、もともと既存にあるソフトウェアを T-Engine 向けに移植したものなどの場合は、必ずしも最初のバージョンが **1.00.00** でなくてもかまわない。