組込み RTOS 向けアプリケーション開発支援ツール
TLV（トレース ログ ヴィジュアライザー）
フェーズ 4
アプリログ拡張外部仕様書

2009 年 6 月 30 日

改訂履歴

版番	日付	更新内容	更新者
1.0	09/6/22	新規作成	柳澤大祐

目次

1 はじめに

1.1 本書の目的

本書の目的は、文部科学省先導的 IT スペシャリスト育成推進プログラム「OJL による最先端技術適応能力を持つ IT 人材育成拠点の形成」プロジェクトにおける、OJL 科目ソフトウェア工学実践研究の研究テーマである「組込み RTOS 向けアプリケーション開発支援ツールの開発」に対して、その開発するソフトウェア拡張の外部仕様を記述することである。

本書は特に、フェーズ 4 におけるアプリログ拡張の外部仕様に関する記述を行う。

1.2 本書の適用範囲

本書は、組込み MPRTOS 向けアプリケーション開発支援ツールの開発プロジェクト（以下本プロジェクト）のフェーズ 4 におけるアプリログ拡張の外部仕様に関する記述を行う。

1.3 用語の定義/略語の説明

表 1　用語定義

用語・略語	定義・説明
TLV	Trace Log Visualizer
MPRTOS	マルチプロセッサ対応リアルタイムオペレーティングシステム
トレースログファイル	RTOS のトレースログ機能を用いて出力したトレースログや、シミュレータなどが出力するトレースログをファイルにしたもの
標準形式トレースログファイル	本ソフトウェアが扱うことの出来る形式をもつトレースログファイル。各種トレースログファイルは、この共通形式トレースログファイルに変換することにより本ソフトウェアで扱うことが出来るようになる。
変換ルール	トレースログファイルを標準形式トレースログファイルに変換する際に用いられるルール。
可視化ルール	標準形式トレースログファイルを可視化する際に用いられるルール。
TLV ファイル	本ソフトウェアが中間形式として用いるファイル。前述の標準形式トレースログファイルは、この TLV ファイルの一部である。

1.4 概要

本書では、組込み MPRTOS 向けアプリケーション開発支援ツールのソフトウェアの仕様を記述する。本書は特に、フェーズ 4 におけるアプリログ拡張の外部仕様に関する記述を行う。

2　概要説明

これまでに寄せられた TLV への要望として、printf デバッグを行いたいというものがあった。一般的な printf デバッグによる出力を TLV のタイムライン上に表示させることによって、メッセージが出力されたときのタスク状態などが一目でわかるという利点がある。この要望を実現するものがアプリログ拡張である。

また、関数の呼び出し関係や列挙型変数の値などを、タスク状態のように可視化したいという要望があったため、これに対してもアプリログ拡張で対応する。

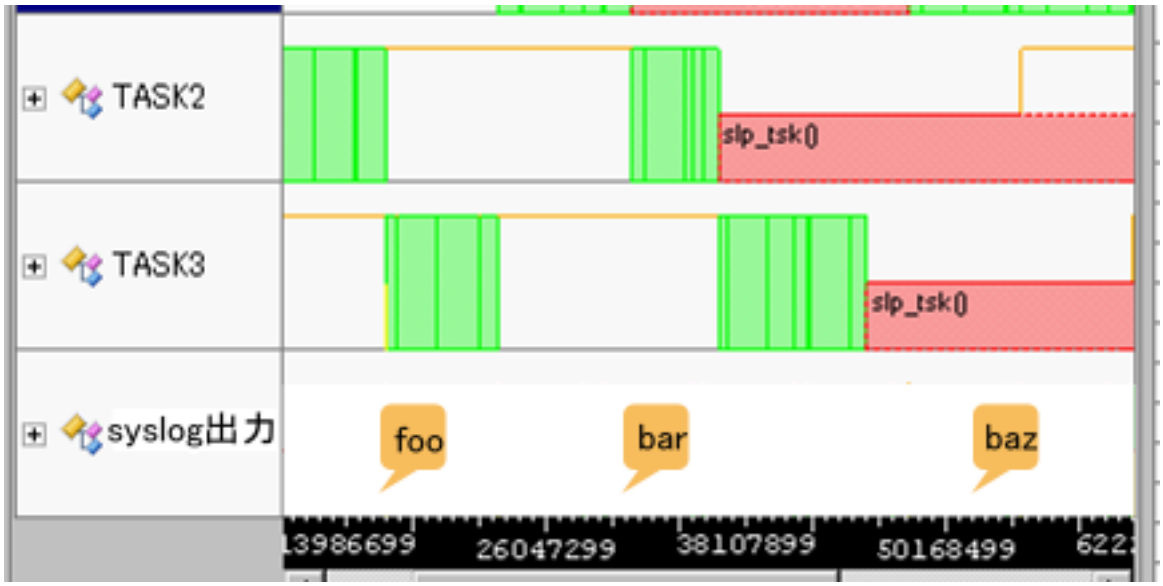

図 1　アプリログ拡張イメージ

2.1　アプリログ拡張

今までの TLV では、任意の文字列を可視化することができないので、printf デバッグを行えない。そこで、フェーズ 4 での要求として任意文字列を可視化することが挙げられている。

例えば、`[123456789] printf foobarbaz.` のようなログがあった場合、時刻 123456789 の位置に foobarbaz というメッセージが表示されるような可視化を想定する。

本拡張では、以下の 3 つの可視化方法を提供する。また、そのそれぞれはタスクに関連するものとしないものの 2 通り存在する。

文字列可視化　任意文字列を TLV 上に表示
ユーザ定義状態可視化　タスクの状態表示のように、ユーザが任意に決めた状態を可視化
端点付きユーザ定義状態可視化　上記の可視化に加え、開始と終了があるもの。関数コールを想定

TOPPERS カーネルでは、ユーザアプリケーションからのログ出力は syslog 関数によって行う。よって、ユーザアプリケーション側からの利用は syslog 関数や、別に用意する API を介して行うものとする。

本機能をユーザアプリケーションから利用する場合は、以下の手順で行う。

1. TLV のリソースファイルに追記
2. アプリケーションにログ出力関数コールを追加（syslog 関数または本拡張で提供する API）
3. アプリケーションを実行
4. TLV を実行

また、今回の拡張は TLV 本体には手を入れず、可視化ルールなどの記述のみで実現することとした。

3 文字列可視化

文字列の可視化には、タスクに関連するものとそうでないものがある。以降、タスクに関連する文字列可視化を**タスク文字列可視化**、タスクに関連しない文字列可視化を**文字列可視化**と呼ぶ。

タスク文字列可視化は、タスクの属性として定義することで、現在のタスク状態表示などの上に重ねて表示する。文字列可視化は、それ専用の行で表示する。

3.1 入力

TLV が認識する方法でログを出力する方法は 2 通りある。ひとつが、TOPPERS カーネルの syslog 関数を呼ぶ方法である。この方法は、指定されたフォーマットで syslog 関数を呼ぶことで、TLV 認識可能なフォーマットのログを出力できる。もうひとつが、本スクリプト拡張で用意する API を利用する方法である。関数の引数に出力したい内容を指定して呼び出して利用する。この方法の場合、syslog 関数のようにフォーマットを気にする必要がない。

斜体で表示されている箇所は、アプリケーション開発者が望む値に置換する場所であることを示している。参考までに、どのようなログが出力されるかを付した。

■書式 文字列可視化をするためのログ出力方法を記述する。

文字列可視化のログ書式

■syslog syslog(”applog str : ID %s : %s.”,*rid*,*str*);

■API void applog_str(const char *_rid_, const char *_str_)

■ログ出力 [*time*] applog str : ID *rid* : *str*.

time 時刻 `[0-9a-zA-Z]+`
rid 表示行 ID `[^.:"]+`
str 出力文字列 `[^.]*`

タスク文字列可視化のログ書式

■syslog syslog(”applog strtask : TASK %d : %s.”,*tid*,*str*);

■API void applog_strtask(const int *tid*, const char *_str_)

■ログ出力 [*time*] applog strtask : TASK *tid* : *str*.

time 時刻 `[0-9a-zA-Z]+`
tid タスク ID `[0-9]+`
str 出力文字列 `[^.]*`

■リソースファイル書式 以下の記述は、文字列可視化を行う際に、ユーザがリソースファイルに追加するものである。タスク文字列可視化では不要である。

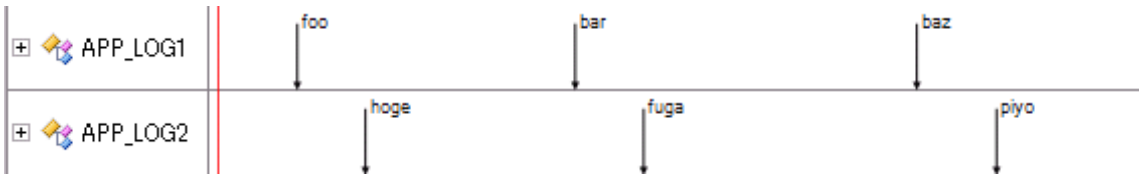

図 2　文字列可視化

文字列可視化リソースファイル書式

```
"rowname":{
    "Type":"ApplogString",
    "Attributes":{
        "id":"rid",
        "str":""
    }
}
```

rowname　文字列可視化行の名前 `[^"]+`
rid　文字列可視化行の ID `[^":]+`

3.2　出力

以下では、文字列の可視化方法を定義する。

3.2.1　文字列可視化

まず、文字列可視化を示す。

```
syslog("applog str : ID 1 : foo.");
```

または

```
applog_str("1", "foo");
```

このようなログ出力および次のようなリソースファイル定義を行った場合、図 2 のように可視化を行う。

リソースファイル

```
"APP_LOG":{
    "Type":"ApplogString",
    "Attributes":{
        "id":"1",
    }
}
```

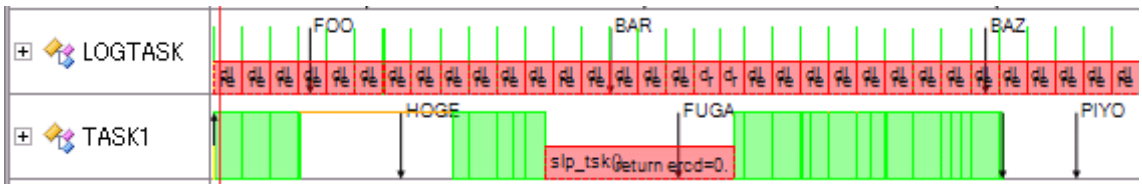

図 3　タスク文字列の可視化

3.2.2　**タスク文字列可視化**

次に、タスクに関連する文字列の可視化を示す。

```
syslog("applog strtask : TASK 1 : FOO.");
```

または

```
applog_strtsk("1", "FOO");
```

このようなログ出力を行った場合、図 3 のように可視化を行う。

タスク文字列の可視化がタスクの状態やシステムコールと重複して見づらい場合は、ユーザが不要な可視化項目を可視化しないようにすることで見やすいよう設定することとする。

4 ユーザ定義状態可視化

以下に、ユーザ定義状態の可視化方法を説明する。TLV は、変換ルール・可視化ルールとしてあらかじめ状態数の定められた状態を提供する。今回は 4 状態および 8 状態の集合とする。ユーザは、可視化したい状態変数の状態数に応じたものを選び、どの状態が何を意味するか決定し、アプリケーションにログ出力コードを埋め込む。

4.1 入力

■書式 ユーザ定義状態可視化をするためのログ出力方法を記述する。

ユーザ定義状態可視化のログ書式

■syslog syslog("applog state id %s %d.",*rid*,*sid*);

■API void applog_stt(const char **rid*, const int *sid*)

■ログ出力 [*time*] applog state id *rid sid*.

time 時刻 `[0-9a-zA-Z]+`
rid 表示行 ID `[^\. ]+`
sid 状態 `[0-9]+`

タスクユーザ定義状態可視化のログ書式

■syslog syslog("applog state task %d %d.",*tid*,*sid*);

■API void applog_stttsk(const int *tid*, const int *sid*)

■ログ出力 [*time*] applog state task *tid sid*.

time 時刻 `[0-9a-zA-Z]+`
tid タスク ID `[0-9]+`
sid 状態 `[0-9]+`

■リソースファイル書式 以下の記述は、ユーザ定義状態可視化を行う際に、ユーザがリソースファイルに追加するものである。タスクユーザ定義状態可視化では不要である。

図 4　ユーザ定義状態可視化

ユーザ定義状態可視化リソースファイル書式（4 状態の場合）

```
"rowname":{
    "Type":"Applog4State",
    "Attributes":{
        "id":"rid",
        "state":sid
    }
}
```

rowname　文字列可視化行の名前 `[^"]+`
rid　文字列可視化行の ID `[^"]+`
sid　初期状態 `[0-9]+`

4.2　出力

以下では、文字列の可視化方法を定義する。

■ユーザ定義状態可視化　まず、ユーザ定義状態可視化を示す。

```
syslog("applog state id 1 0.");
syslog("applog state id 1 1.");
syslog("applog state id 1 2.");
syslog("applog state id 1 1.");
syslog("applog state id 1 0.");
```

または

```
applog_stt("1", 0);
applog_stt("1", 1);
applog_stt("1", 2);
applog_stt("1", 1);
applog_stt("1", 0);
```

このようなログ出力を行った場合、図 4 のように可視化を行う。

5　ユーザ定義状態可視化（端点付き）

以下に、端点付きのユーザ定義状態の可視化方法を説明する。通常との違いは、状態の開始時と終了時に端点がついている点である。これにより関数コールなどを表現する。

5.1　入力

■書式　端点付きユーザ定義状態可視化をするためのログ出力方法を記述する。

端点付きユーザ定義状態可視化のログ書式

■syslog

syslog(”applog state id %s %d start.”,*rid*,*sid*);
syslog(”applog state id %s %d end.”,*rid*,*sid*);

■API

void applog_stt_st(const char ***rid*, const int *sid*)
void applog_stt_en(const char ***rid*, const int *sid*)

■ログ出力

[*time*] applog state id *rid sid* start.
[*time*] applog state id *rid sid* end.

time　時刻 `[0-9a-zA-Z]+`
rid　表示行 ID `[^\. ]+`
sid　状態 `[0-9]+`

端点付きタスクユーザ定義状態可視化のログ書式

■syslog

syslog(”applog state task %s %d start.”,*tid*,*sid*);
syslog(”applog state task %s %d end.”,*tid*,*sid*);

■API

void applog_stttsk_st(const char ***tid*, const int *sid*)
void applog_stttsk_en(const char ***tid*, const int *sid*)

■ログ出力

[*time*] applog state tsk *tid sid* start.
[*time*] applog state tsk *tid sid* end.

time　時刻 `[0-9a-zA-Z]+`
tid　タスク ID `[0-9]+`
sid　状態 `[0-9]+`

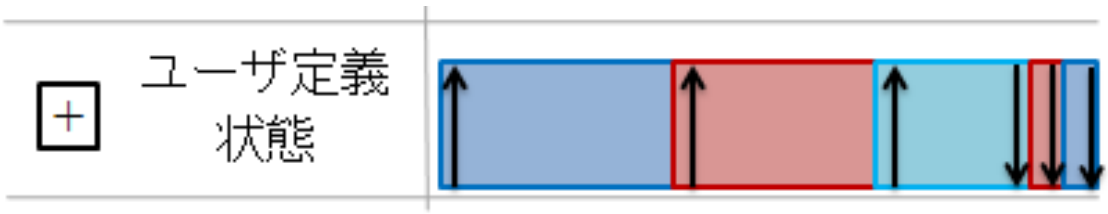

図 5　ユーザ定義状態可視化

■リソースファイル書式　以下の記述は、端点付きユーザ定義状態可視化を行う際に、ユーザがリソースファイルに追加するものである。端点付きタスクユーザ定義状態可視化では不要である。

ユーザ定義状態可視化リソースファイル書式（4 状態の場合）

```
"rowname":{
    "Type":"Applog4StateTerminal",
    "Attributes":{
        "id":"rid",
        "state":sid
    }
}
```

rowname　文字列可視化行の名前 [^"]+
rid　文字列可視化行の ID [^"]+
sid　初期状態 [0-9]+

5.2　出力

以下では、端点付きユーザ定義状態の可視化方法を定義する。端点付きタスクユーザ定義状態に関しては省略する。

```
syslog("applog state id 1 0 start.");
syslog("applog state id 1 1 start.");
syslog("applog state id 1 2 start.");
syslog("applog state id 1 2 end.");
syslog("applog state id 1 1 end.");
syslog("applog state id 1 0 end.");
```

または

```
applog_stt_st("1", 0);
applog_stt_st("1", 1);
applog_stt_st("1", 2);
```

```
applog_stt_en("1", 2);
applog_stt_en("1", 1);
applog_stt_en("1", 0);
```

このようなログ出力を行った場合、図 5 のように可視化を行う。

参考文献