最終更新日 ： 2012/02/22

TOPPERS/ASPカーネル
標準トレースログの可視化変換ルール例

名古屋大学

はじめに

❖本ドキュメントは，可視化変換ルールについて，
TOPPERS/ASPカーネルのトレースログを例に解説する

目次

- ❖ 可視化変換ルールのマニュアル
  - ファイル構成
  - ファイル説明
  - 可視化変換ルールとは？
  - 可視化表示の仕組み
- ❖ 可視化変換の流れ
  - 図形データの生成（例）
- ❖ 可視化ルールの作成
  - 図形の定義
  - 図形の定義のメンバ
  - 可視化ルールの定義
  - 可視化ルールの定義のメンバ
  - 可視化ルールの作成（例）
- ❖ 周期ハンドラの可視化(例)
  - ・周期ハンドラの属性分析.
  - ・周期ハンドラの希望出力図形を描いてみる.
  - ・出力されているトレースログ確保.
  - ・周期ハンドラの図形の定義.
  - ・周期ハンドラの可視化ルールの定義.
  - ・出力される図形の確認.

可視化変換ルールのマニュアル

ファイル構成

❖TLV内の関連ファイル

convertRules > asp.cnv, fmp.cnv

resourceHeaders > asp.resh, fmp.resh

visualizeRules > asp_rules.viz, asp_shapes.viz,
fmp_rules.viz, fmp_shapes.viz,
toppers_rules.viz, toppers_shapes.viz

❖ASPカーネル内の関連ファイル

- asp/kernel/kernel.tf
- asp/arch/logtrace/tlv.tf

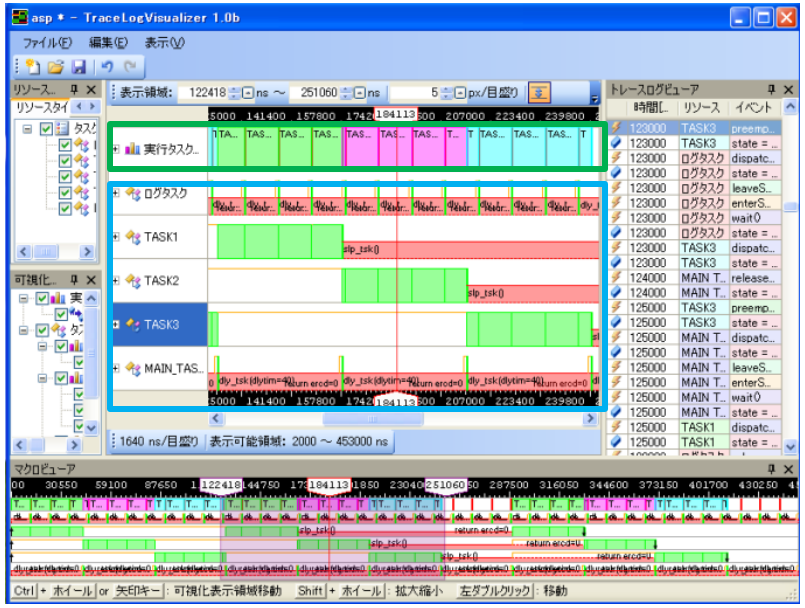
ファイル説明

❖実行タスク変化を表すための可視化ルール(ASP)

- asp_rules.viz：可視化ルールの定義
- asp_shapes.viz ：図形の定義

❖リソースの変化を表すための可視化ルール(共通)

- toppers_rules.viz：可視化ルールの定義
- toppers_shapes.viz：図形の定義



今回の説明対象

可視化変換ルールとは？

❖可視化表示項目毎に要求される表現を，トレースログの内容に従い時系列の図として画面に描画する処理.

ASPカーネルのトレースログ

```
 1 [2000] dispatch to task 1.
 2 [2000] enter to ena_int intno=2.
 3 [2000] leave to ena_int ercd=0.
 4 [2000] enter to dly_tsk dlytim=10.
 5 [2000] task 1 becomes WAITING.
 6 [2000] dispatch to task 5.
 7 [2000] enter to act_tsk tskid=2.
 8 [2000] task 2 becomes RUNNABLE.
 9 [2000] leave to act_tsk ercd=0.
10 [2000] enter to act_tsk tskid=3.
11 [2000] task 3 becomes RUNNABLE.
12 [2000] leave to act_tsk ercd=0.
13 [2000] enter to act_tsk tskid=4.
14 [2000] task 4 becomes RUNNABLE.
15 [2000] leave to act_tsk ercd=0.
16 [2000] enter to dly_tsk dlytim=40.
```

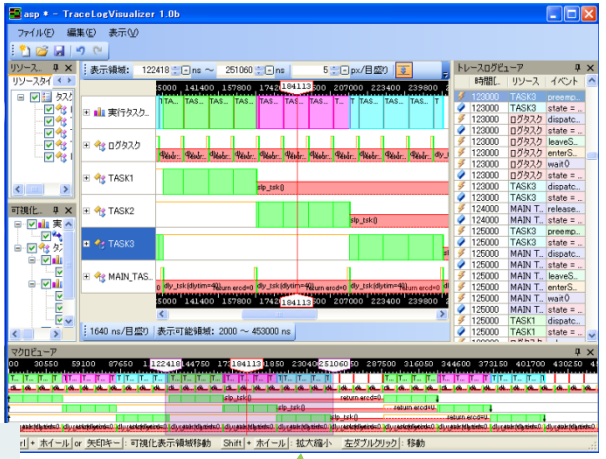
標準形式変換ルールで，
標準形式に変換.

```
 2 ]LOGTASK.dispatch()",
 3 ]LOGTASK.state=RUNNING",
 4 ]LOGTASK.enterSVC(ena_int,intno=2)",
 5 ]LOGTASK.leaveSVC(ena_int,ercd=0)",
 6 ]LOGTASK.enterSVC(dly_tsk,dlytim=10)",
 7 ]LOGTASK.wait()",
 8 ]LOGTASK.state=WAITING",
 9 ]MAIN_TASK.dispatch()",
10 ]MAIN_TASK.state=RUNNING",
11 ]MAIN_TASK.enterSVC(act_tsk,tskid=2)",
12 ]TASK1.activate()",
13 ]TASK1.state=RUNNABLE",
14 ]MAIN_TASK.leaveSVC(act_tsk,ercd=0)",
15 ]MAIN_TASK.enterSVC(act_tsk,tskid=3)",
16 ]TASK2.activate()",
```

標準形式トレースログ

今回は，可視化変換ルールについて説明

可視化表示



可視化変換ルールで，
図形データ生成

可視化表示の仕組み

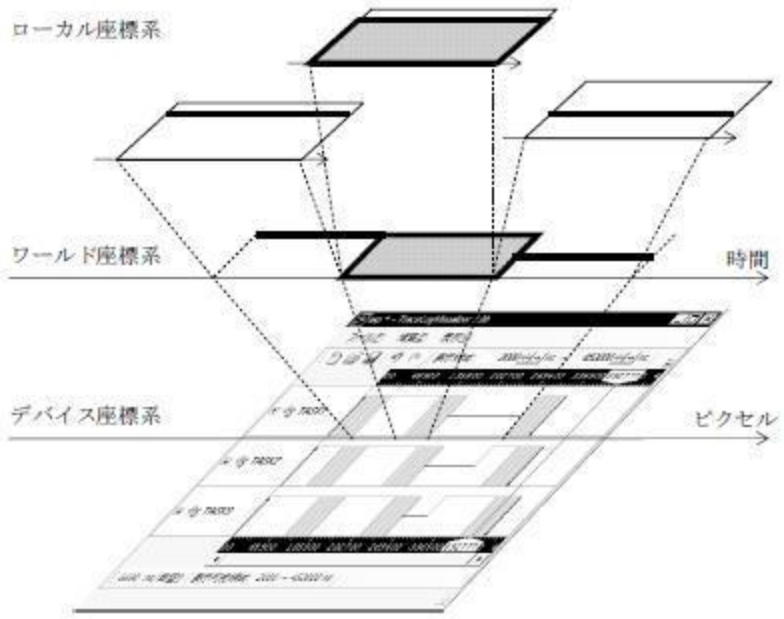


図　抽象的な可視化表示の仕組みの例

❖図形データの生成とは，標準形式トレースログに対して可視化ルールを適用し，ローカル座標系に図形データを生成，ワールド座標系に図形を割り当てる処理.

❖ワールド座標系へ割り当てられた図形は，表示開始時刻と単位時間あたりのピクセル数により表示デバイスに表示.

可視化表示の仕組み

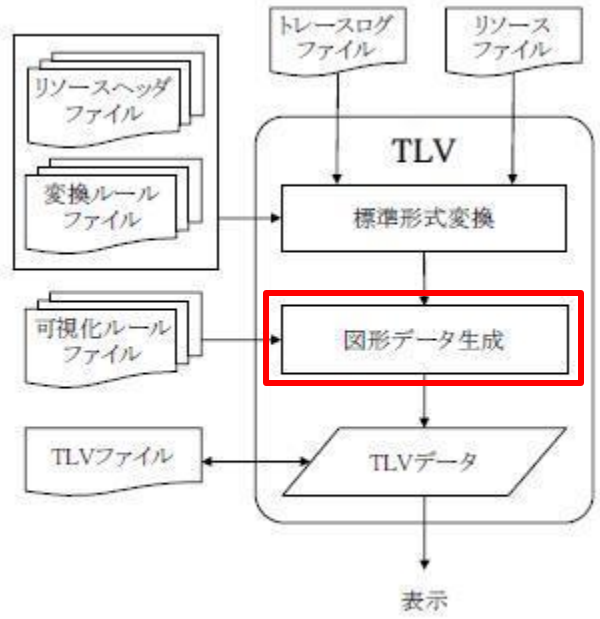


図　TLV の構成

- ❖図形データの生成は，図形の定義と可視化ルールの定義に従い，標準形式トレースログに可視化ルールを適用することで行う．図形と可視化ルールの定義は可視化ルールファイルに記述する．

- ❖この処理には外部ファイルとして，可視化ルールファイルがリソースファイルの定義に従い読み込まれる．

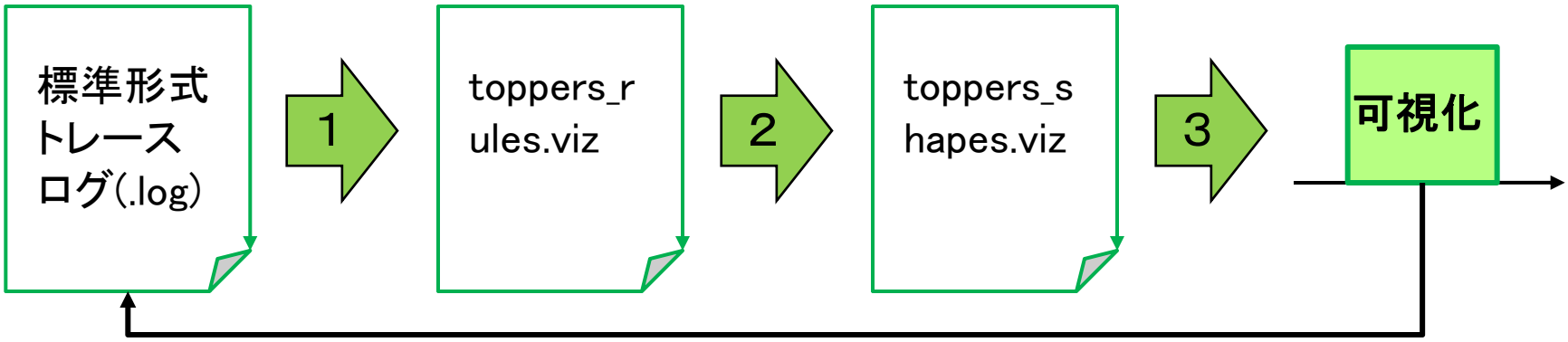
可視化変換の流れ

可視化変換の流れ

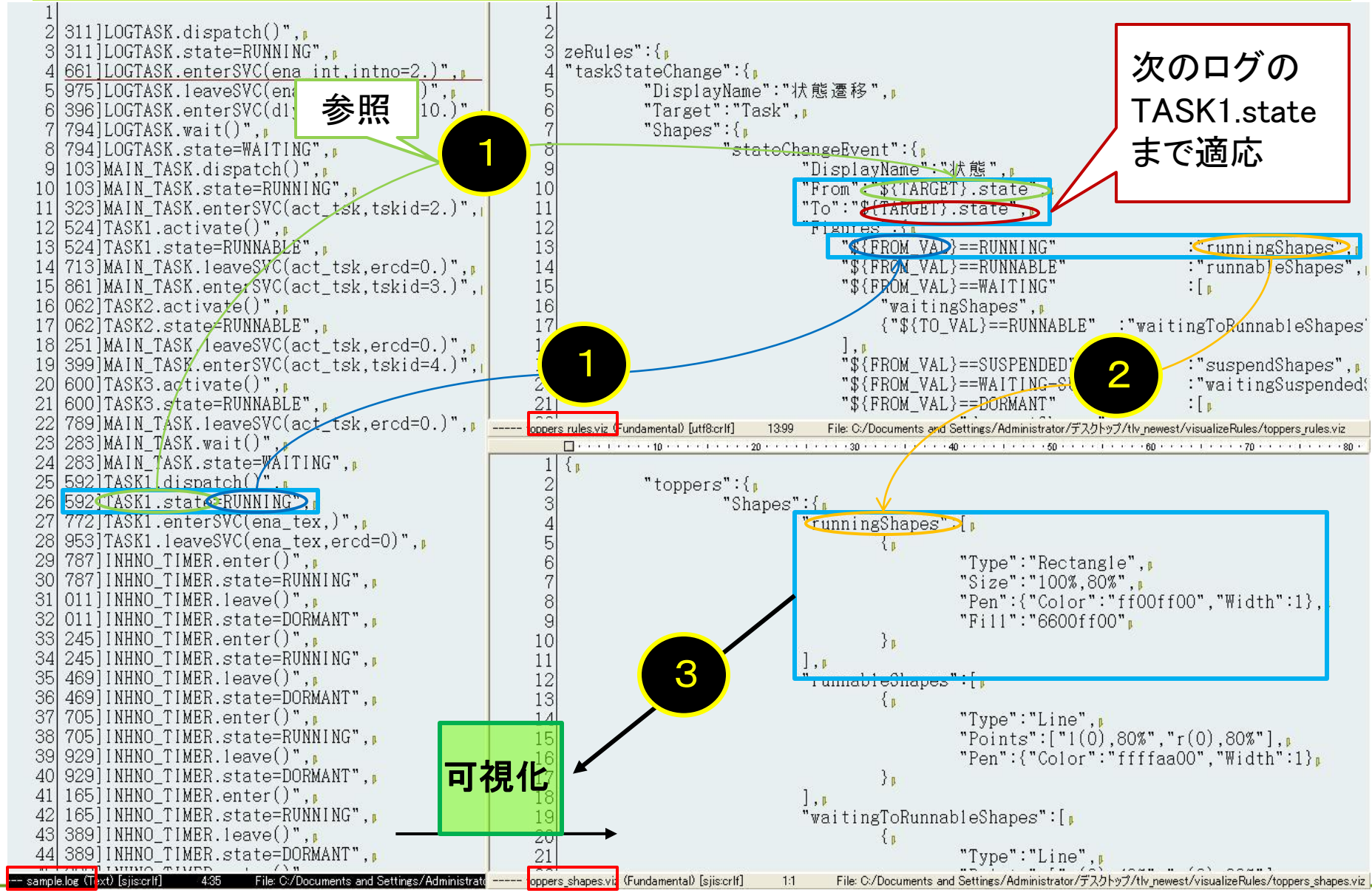
❖処理順序は次の通りです.

1．標準形式に変換されたトレースログを可視化ルールの定義(toppers_rules.viz)によって，時間順で読み込む.
2．図形の定義(toppers_shapes.viz)で，トレースログの内容から可視化表現を決定.
3．決定された各ログの図形を出力.

❖上記の処理を標準形式トレースログの最後まで繰り返す.



図形データの生成（例）

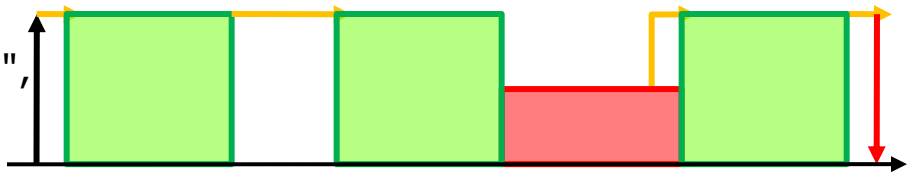

参照
1
1
2
3
次のログの
TASK1.state
まで適応
可視化
"taskStateChange":{
"DisplayName":"状態遷移",
"Target":"Task",
"Shapes":{
"stateChangeEvent":{
"DisplayName":"状態",
"From":"${TARGET}.state",
"To":"${TARGET}.state",
"Figures":[
"${FROM_VAL}==RUNNING" :"runningShapes",
"${FROM_VAL}==RUNNABLE" :"runnableShapes",
"${FROM_VAL}==WAITING" :[
"waitingShapes",
{"${TO_VAL}==RUNNABLE" :"waitingToRunnableShapes"
],
"${FROM_VAL}==SUSPENDED" :"suspendShapes",
"${FROM_VAL}==WAITING-S :"waitingSuspended
"${FROM_VAL}==DORMANT" :[
"toppers":{
"Shapes":{
"runningShapes":[
{
"Type":"Rectangle",
"Size":"100%,80%",
"Pen":{"Color":"ff00ff00","Width":1},
"Fill":"6600ff00"
}
],
"runnableShapes":[
{
"Type":"Line",
"Points":["1(0),80%","r(0),80%"],
"Pen":{"Color":"ffffaa00","Width":1}
}
],
"waitingToRunnableShapes":[
{
"Type":"Line",
"sample.log (Text) [sjis:crlf] 4:35 File: C:/Documents and Settings/Administrat
oppers_rules.viz (Fundamental) [utf8:crlf] 13:99 File: C:/Documents and Settings/Administrator/デスクトップ/tlv_newest/visualizeRules/toppers_rules.viz
oppers_shapes.viz (Fundamental) [sjis:crlf] 1:1 File: C:/Documents and Settings/Administrator/デスクトップ/tlv_newest/visualizeRules/toppers_shapes.viz

図形データの生成（例）

- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.dispatch()",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.preempt()",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.dispatch()",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.wait()",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.releaseFromWaiting()",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.dispatch()",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.preempt()",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

- 今回は左の標準形式トレースログファイル(.log)の処理を見ていきます.

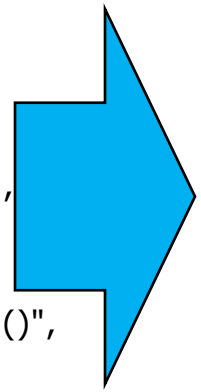
- 以下のような図形が出力されます.



図形データの生成（例）

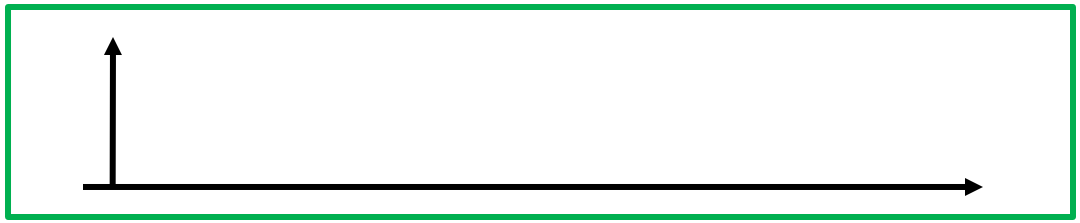
- 理解しやすくするために，図形の可視化に係るログだけを抽出して説明します（実際の処理と関係なし ）

- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.dispatch()",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.preempt()",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.dispatch()",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.wait()",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.releaseFromWaiting()",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.dispatch()",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.preempt()",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",



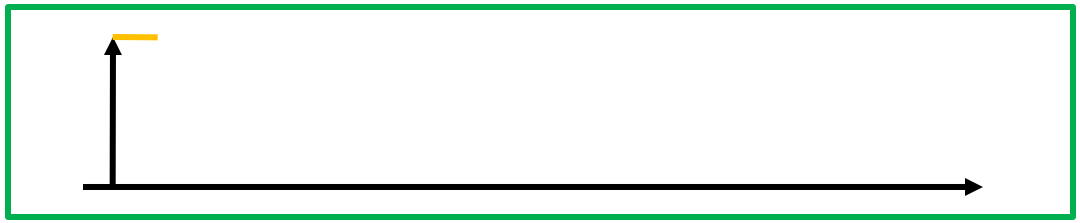
- **"[1594524]TASK1.activate()",**
- **"[1594524]TASK1.state=RUNNABLE",**
- **"[1596592]TASK1.state=RUNNING",**
- **"[14588885]TASK1.state=RUNNABLE",**
- **"[20737820]TASK1.state=RUNNING",**
- **"[27407562]TASK1.enterSVC(slp_tsk,)",**
- **"[27407811]TASK1.state=WAITING",**
- **"[35190794]TASK1.state=RUNNABLE",**
- **"[35191803]TASK1.state=RUNNING",**
- **"[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",**
- **"[45376542]TASK1.state=RUNNABLE",**
- **"[45377586]TASK1.terminate()",**
- **"[45377586]TASK1.state=DORMANT",**

図形データの生成（例）



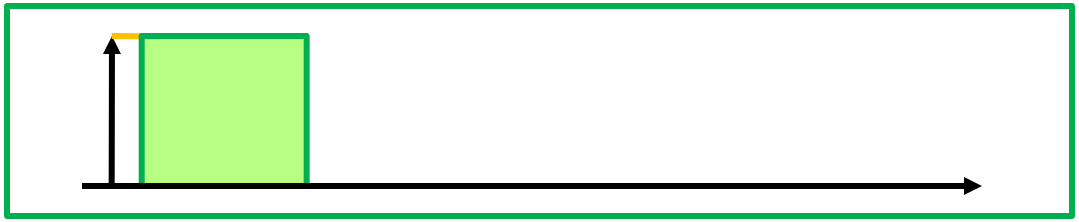
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



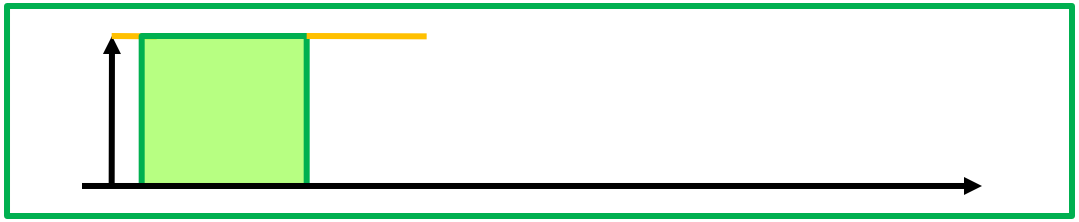
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



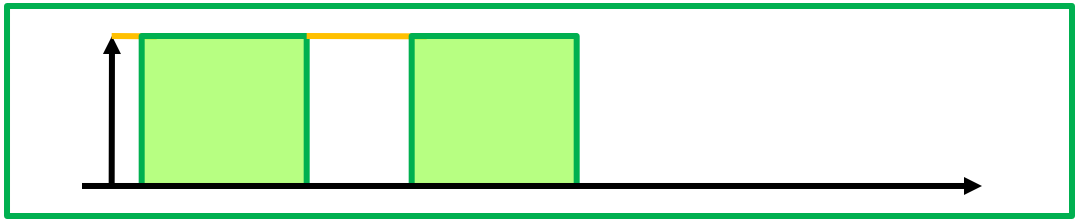
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



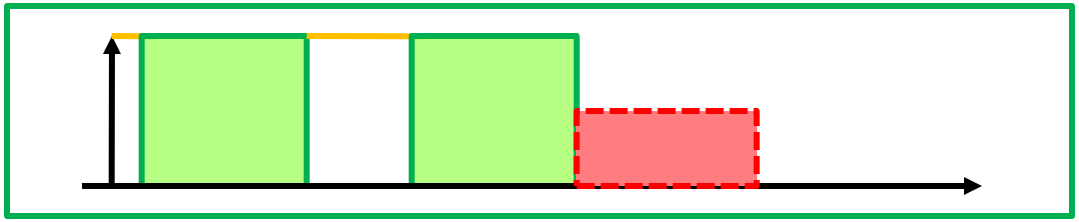
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



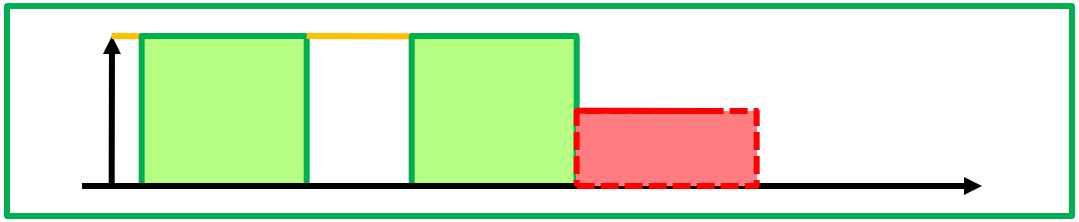
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



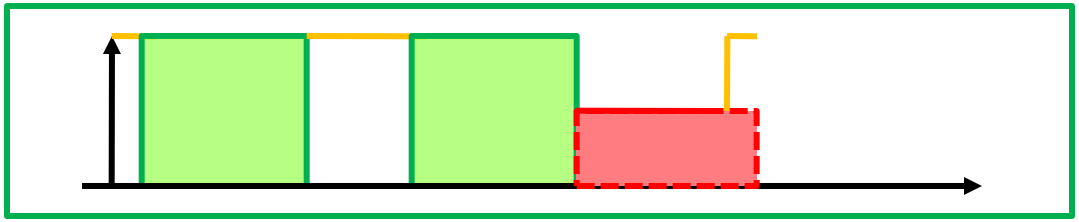
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



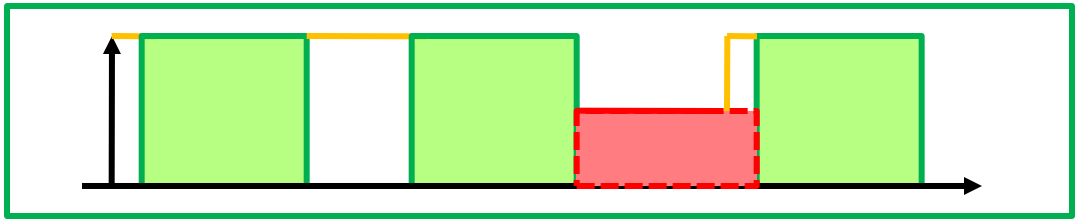
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



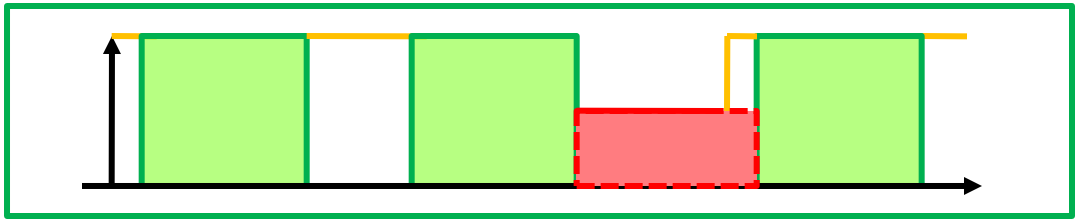
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



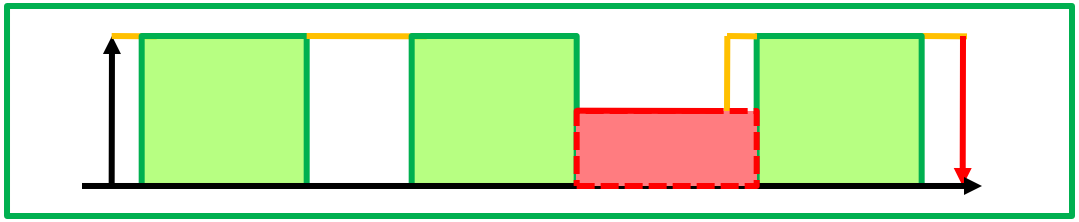
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



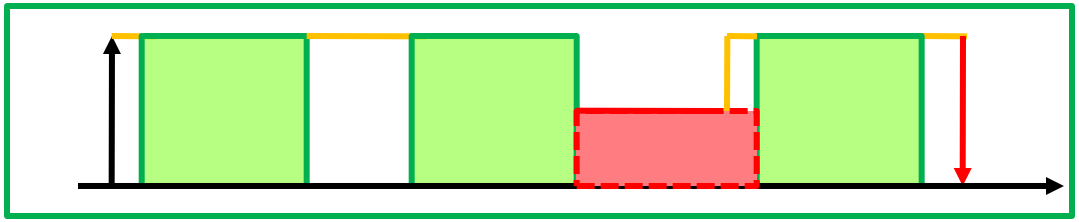
- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

図形データの生成（例）



❖"[1594524]TASK1.activate()",
❖"[1594524]TASK1.state=RUNNABLE",
❖"[1596592]TASK1.state=RUNNING",
❖"[14588885]TASK1.state=RUNNABLE",
❖"[20737820]TASK1.state=RUNNING",
❖"[27407562]TASK1.enterSVC(slp_tsk,)",
❖"[27407811]TASK1.state=WAITING",
❖"[35190794]TASK1.state=RUNNABLE",
❖"[35191803]TASK1.state=RUNNING",
❖"[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
❖"[45376542]TASK1.state=RUNNABLE",
❖"[45377586]TASK1.terminate()",
❖"[45377586]TASK1.state=DORMANT",

図形データの生成（例）



- "[1594524]TASK1.activate()",
- "[1594524]TASK1.state=RUNNABLE",
- "[1596592]TASK1.state=RUNNING",
- "[14588885]TASK1.state=RUNNABLE",
- "[20737820]TASK1.state=RUNNING",
- "[27407562]TASK1.enterSVC(slp_tsk,)",
- "[27407811]TASK1.state=WAITING",
- "[35190794]TASK1.state=RUNNABLE",
- "[35191803]TASK1.state=RUNNING",
- "[35191974]TASK1.leaveSVC(slp_tsk,ercd=0.)",
- "[45376542]TASK1.state=RUNNABLE",
- "[45377586]TASK1.terminate()",
- "[45377586]TASK1.state=DORMANT",

可視化ルールの作成

可視化ルールの作成

❖可視化ルールの作成には，大きく二つの過程があります．

１．図形の定義

２．可視化ルールの定義

図形の定義

❖図形の定義とは，抽象化した図形を形式化したものです．

❖図形の定義は“Shapes”というメンバ名の値にオブジェクトとして記述します．このオブジェクトのメンバ名には図形の名前を記述します．そして，その値に図形の定義を基本図形の定義の配列として与えます．

図形の定義のメンバ

❖基本図形の定義に用いるメンバは，基本図形の形状により異なります.

❖基本図形として，”Rectangle”：長方形，”Line”：線分，”Arrow”：矢印，”Polygon”：多角形，”Pie”：扇形，”Ellipse”：楕円形，”Text”：文字列のいずれかを使えます.

❖今回は，以下の四つについて説明します.

・Rectangle ・Line ・Arrow ・Text

順番に見ていきます.

❖まずは，Rectangleです.

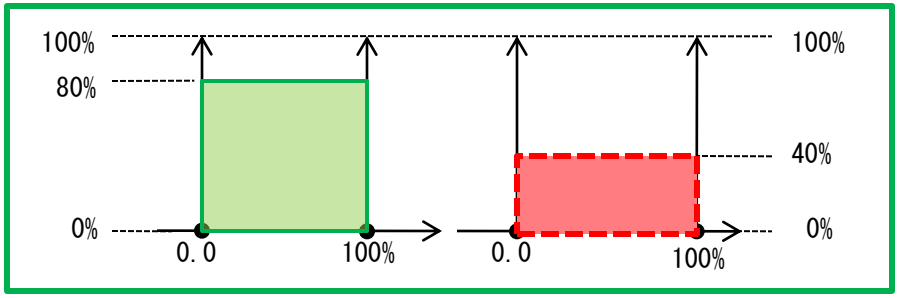
図形の定義 – Type : Rectangle

❖以下のような属性を使うことができます.

❖Type：図形の形状.
❖Location：図形の位置.
❖Size ：図形のサイズ.
❖Pen ：縁取り線を指定する.
❖Fill ：塗りつぶしの色.
❖Alpha ：塗りつぶしの色の透明度.

❖DashStyleも使用可能.

図形の定義 - Type : Rectangle




```
"runningShapes":[
            {
            "Type":"Rectangle",
            "Size":"100%,80%",
            "Pen":{"Color":"ff00ff00","Width":1},
            "Fill":"6600ff00"
            }
   ]
```

```
"svcShapes":[
            {
            "Type":"Rectangle",
            "Size":"100%,40%",
            "Pen":{"Color":"${ARG0}","Width":1,
            "DashStyle":"Dash"},
            "Fill":"${ARG0}",
            "Alpha":100
            }
]
```

図形の定義 – Type : Line/Arrow

❖ “Line/Arrow”は以下のような属性を使うことができます.

❖Points：始点と終点を指定する.

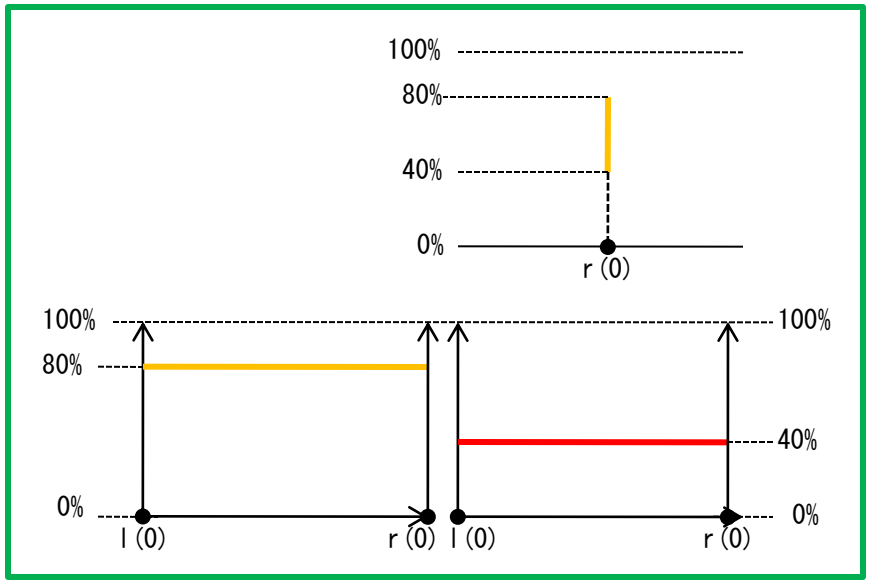
❖Pen：縁取り線を指定する.

値はオブジェクト. メンバとして以下が使える

- Color - 線の色
- Alpha – 線の透明度
- Width - 線の幅

❖Type,Size,Location,Fill,Alpha,DashStyle,Arc(扇形の開始角度)も使用可能.

図形の定義 - Type : Line

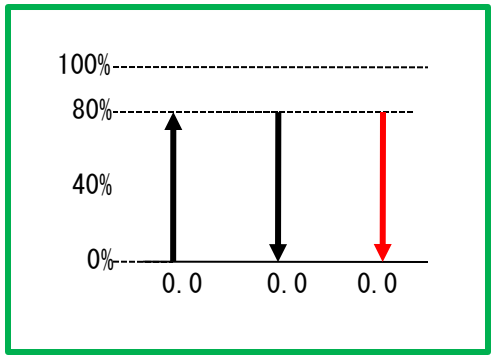



```
"waitingToRunnableShapes":[
        {
        "Type":"Line",
        "Points":["r(0),40%","r(0),80%"],
        "Pen":{"Color":"ffffaa00","Width":1}
        }
]
```

```
"runnableShapes":[
        {
        "Type":"Line",
        "Points":["l(0),80%","r(0),80%"],
        "Pen":{"Color":"ffffaa00","Width":1}
        }
]
```

```
"waitingShapes":[
        {
        "Type":"Line",
        "Points":["l(0),40%","r(0),40%"],
        "Pen":{"Color":"ffff0000","Width":1}
        }
]
```

図形の定義 - Type：Arrow




```
"activateShapes":[
        {
        "Type":"Arrow",
        "Points":["0,0","0,80%"],
        "Pen":{"Color":"ff000000","Width":1}
        }
]
```

```
"terminateShapes":[
        {
        "Type":"Arrow",
        "Points":["0,80%","0,0"],
        "Pen":{"Color":"ffff0000","Width":1}
        }
]
```

```
"exitShapes":[
        {
        "Type":"Arrow",
        "Points":["0,80%","0,0"],
        "Pen":{"Color":"ff000000","Width":1}
        }
]
```

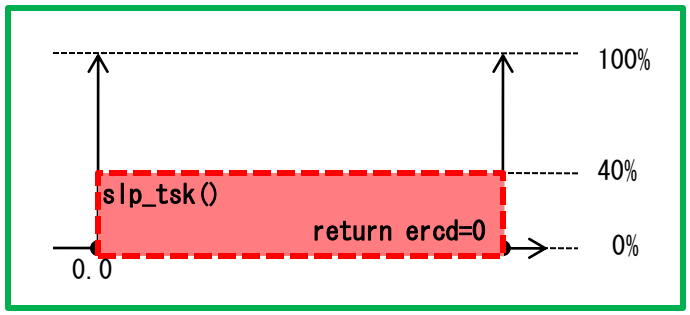
図形の定義 – Type : Text

❖ “Text ”は以下のような属性を使うことができます.

❖Size：図形のサイズ.
❖Text：描画する文字列を指定する
❖Font：描画する文字列の色，透明度，フォント，サイズ，
スタイル，領域内での位置を指定する.
オブジェクトのメンバとして以下が使える.
- Align：文字列の領域内での位置
- Size：文字列のサイズ
- Color，Family，Style，Alphaも使用可能.

❖Type,Locationも使用可能.

図形の定義 - Type : Text



* ${ARG1} = “slp_tsk()”
* ${ARG2} = “ercd = 0”

```
"svcShapes":[
        {
        "Type":"Rectangle",
        "Size":"100%,40%",
        "Pen":{"Color":"${ARG0}","Width":1,
        "Alpha":255, "DashStyle":"Dash"},
        "Fill":"${ARG0}",
        "Alpha":100
        },
        {
        "Type":"Text",
        "Size":"100%,40%",
        "Font":{"Align":"TopLeft", "Size":7},
        "Text":"${ARG1}"
        },
        {
        "Type":"Text",
        "Size":"100%,40%",
        "Font":{"Align":"BottomRight","Size":7},
        "Text":"return ${ARG2}"
        }
]
```

可視化ルールの作成

❖可視化ルールの作成には，大きく二つの過程があります．

１．図形の定義

２．可視化ルールの定義

❖次は，可視化ルールの定義について説明したいと思います．

可視化ルールの定義

- 可視化ルールの定義とは，可視化ルールを形式化したものです．

- 可視化ルールの定義は“VisualizeRules”というメンバ名の値にオブジェクトとして記述します．このオブジェクトのメンバ名には可視化ルールの名前を記述します．そして，その値に可視化ルールの定義を記述します．

可視化ルールの定義のメンバ

❖可視化ルールの定義に用いるメンバはDisplayName，Target，Shapesの三種類です．

❖DisplayName：可視化ルールの表示名
❖Target：可視化ルールを適用するリソースタイプ
❖Shapes：図形群の定義を記述する

メンバ名に図形群の名前，メンバの値として図形群の定義をオブジェクトとして与える．その際，以下のメンバが使えます．(次のスライド)

可視化ルールの定義 – Shapesのメンバ

・DisplayName：図形群の表示名
・From ：図形群を構成する図形のワールド変換に適用される開始イベント
・To ：図形群を構成する図形のワールド変換に適用される終了イベント
・When：図形群を構成する図形のワールド変換に適用されるイベント．開始時刻と終了時刻が同じ場合に用いる
・Figures：図形群を構成する図形の定義

可視化ルールの作成(例)

```
"DisplayName":"状態遷移",
   "Target":"Task",
          "Shapes":{
          "stateChangeEvent":{
                    "DisplayName":"状態",
                    "From":"${TARGET}.state",
                    "To":"${TARGET}.state",
                    "Figures":{
                        "${FROM_VAL}==RUNNING"              :"runningShapes",
                        "${FROM_VAL}==RUNNABLE"             :"runnableShapes",
                        "${FROM_VAL}==WAITING"              :[
                            "waitingShapes",
                            {"${TO_VAL}==RUNNABLE"  :"waitingToRunnableShapes"}
                        ],
                        "${FROM_VAL}==SUSPENDED"            :"suspendShapes",
                        "${FROM_VAL}==WAITING-SUSPENDED"    :"waitingSuspendedShapes",
                        "${FROM_VAL}==DORMANT"              :[
                            "dormantShapes",
                            {"${TO_VAL}==RUNNABLE"  :"dormantToRunnableShapes"}
                        ]
                    }
          },
```

可視化ルールの作成(例)

```
"callSvc":{
   "DisplayName":"システムコール",
   "Target":"Task",
   "Shapes":{
          "callSvcEvent":{
                   "DisplayName":"システムコール",
                   "From":"${TARGET}.enterSVC()",
                   "To":"${TARGET}.leaveSVC(${FROM_ARG0})",
                   "Figures":[
                    {"${FROM_ARG0}==slp_tsk||${FROM_ARG0}==dly_tsk"
                             :"svcShapes(ff0000,${FROM_ARG0}(${FROM_ARG1}),${TO_ARG1})"},
                    {"${FROM_ARG0}!=slp_tsk&&${FROM_ARG0}!=dly_tsk"
                             :"svcShapes(ffff00,${FROM_ARG0}(${FROM_ARG1}),${TO_ARG1})"}
                   ]
          }
}
```

周期ハンドラの可視化

周期ハンドラの可視化順序

1. 周期ハンドラの属性分析.
2. 周期ハンドラの可視化表現の要求を定める.
3. 周期ハンドラのログを含むトレースログの取得.
4. 周期ハンドラの図形の定義.
5. 周期ハンドラの可視化ルールの定義.
6. 出力される図形の確認.

1．周期ハンドラの属性分析

参考
TOPPERS新世代カーネル
統合仕様書1.3.0より

❖まず，周期ハンドラの属性から知る必要があります．

CRE_CYC 周期ハンドラの生成〔S〕
acre_cyc 周期ハンドラの生成〔TD〕

【静的API】
CRE_CYC(ID cycid, { ATR cycatr, intptr_t exinf, CYCHDR cychdr, RELTIM cyctim, RELTIM cycphs })

【C言語API】
ER_ID cycid = acre_cyc(const T_CCYC *pk_ccyc)

【パラメータ】
ID cycid 生成する周期ハンドラのID番号（acre_cycを除く）
T_CCYC * pk_ccyc 周期ハンドラの生成情報を入れたパケットへのポインタ（静的APIを除く）
＊周期ハンドラの生成情報（パケットの内容）
ATR cycatr 周期ハンドラ属性
intptr_t exinf 周期ハンドラの拡張情報
CYCHDR cychdr 周期ハンドラの先頭番地
RELTIM cyctim 周期ハンドラの起動周期
RELTIM cycphs 周期ハンドラの起動位相

【リターンパラメータ】
ER_ID cycid 生成された周期ハンドラのID番号（正の値）またはエラーコード

【エラーコード】
E_CTX コンテキストエラー（非タスクコンテキストからの呼出し，CPUロック状態からの呼出し）
E_RSATR 予約属性（cycatrが不正または使用できない）
E_PAR パラメータエラー（cychdr，cyctim，cycphsが不正）
E_OACV〔P〕オブジェクトアクセス違反（システム状態に対する管理操作が許可されていない）
E_MACV〔P〕メモリアクセス違反（pk_ccycが指すメモリ領域への読出しアクセスが許可されていない）
E_NOID ID番号不足（割り付けられる周期ハンドラIDがない：acre_cycの場合）
E_OBJ オブジェクト状態エラー（cycidで指定した周期ハンドラが登録済み：acre_cycを除く）

【機能】
各パラメータで指定した周期ハンドラ生成情報に従って，周期ハンドラを生成する．具体的な振舞いは以下の通り．cycatrにTA_STAを指定した場合，対象周期ハンドラは動作している状態となる．次に周期ハンドラを起動するシステム時刻は，静的APIの場合はcycphsで指定したシステム時刻に，サービスコールの場合は呼び出してからcycphsで指定した相対時間後に設定される．cycatrにTA_STAを指定しない場合，対象周期ハンドラは動作していない状態に初期化される．静的APIにおいては，cycidはオブジェクト識別名，cycatr，cyctim，cycphsは整数定数式パラメータ，exinfとcychdrは一般定数式パラメータである．cyctimは，0より大きく，TMAX_RELTIM以下の値でなければならない．また，cycphsは，TMAX_RELTIM以下でなければならない．cycphsにcyctimより大きい値を指定してもよい．

【補足説明】
静的APIにおいて，cycatrにTA_STAを，cycphsに0を指定した場合，周期ハンドラが最初に呼び出されるのは，カーネル起動後最初のタイムティックになる．cycphsに1を指定した場合も同じ振舞いとなるため，静的APIでcycphsに0を指定することは推奨しないこととし，コンフィギュレータが警告メッセージを出力する．

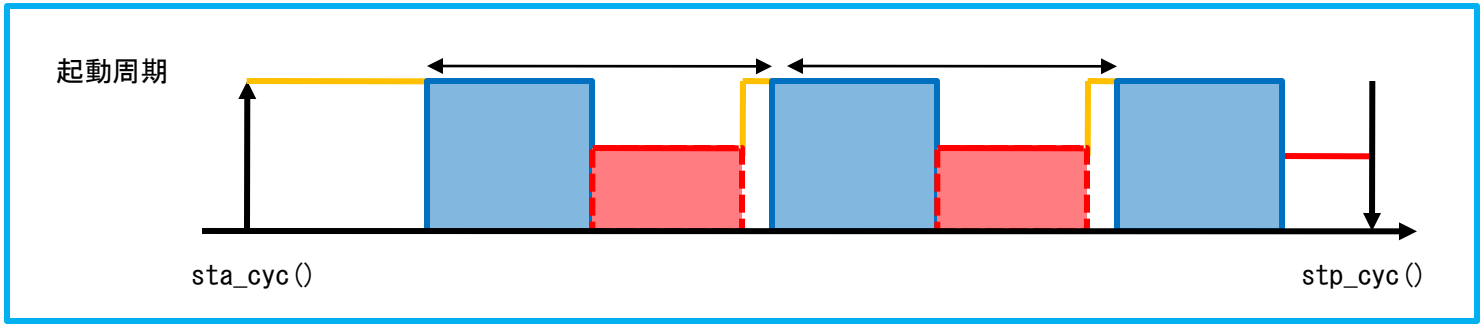
【TOPPERS/ASPカーネルにおける規定】
ASPカーネルでは，CRE_CYCのみをサポートする．ただし，TA_PHS属性の周期ハンドラはサポートしない．

１．周期ハンドラの属性分析

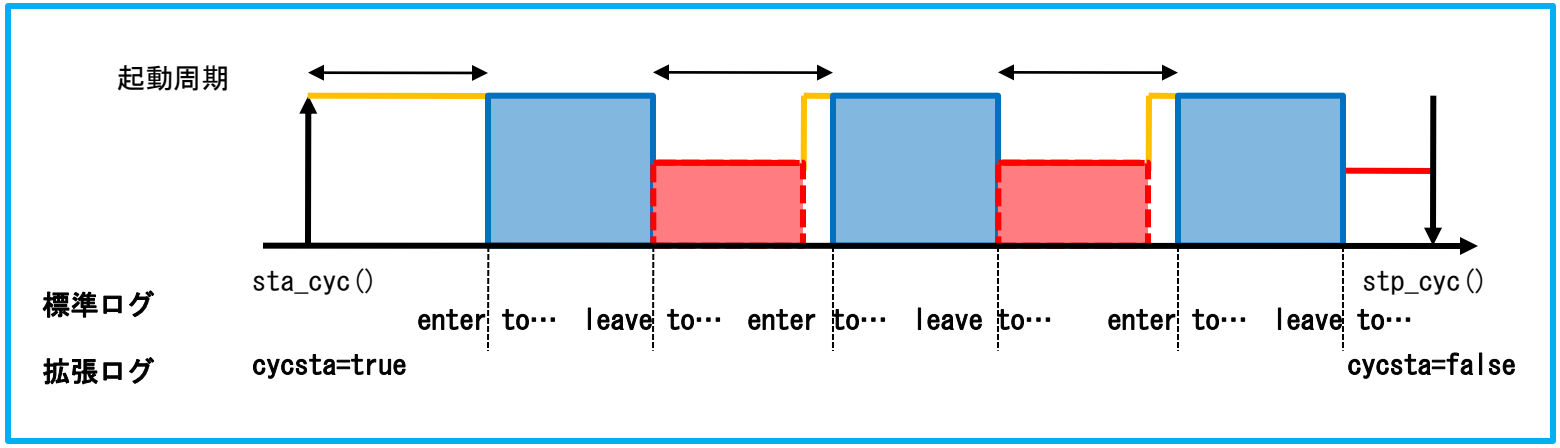
- ❖仕様書のどの部分を参考にして可視化表現の要求を定めるのか，記述する．

２．周期ハンドラの可視化表現の要求を定める．

❖希望出力図形



❖上記のような図形を出力するためには，以下のような周期ハンドラのトレースログが必要される．



３．周期ハンドラのログを含むトレースログの取得.

❖現在，出力されている周期ハンドラのトレースログは標準トレースログのみ.

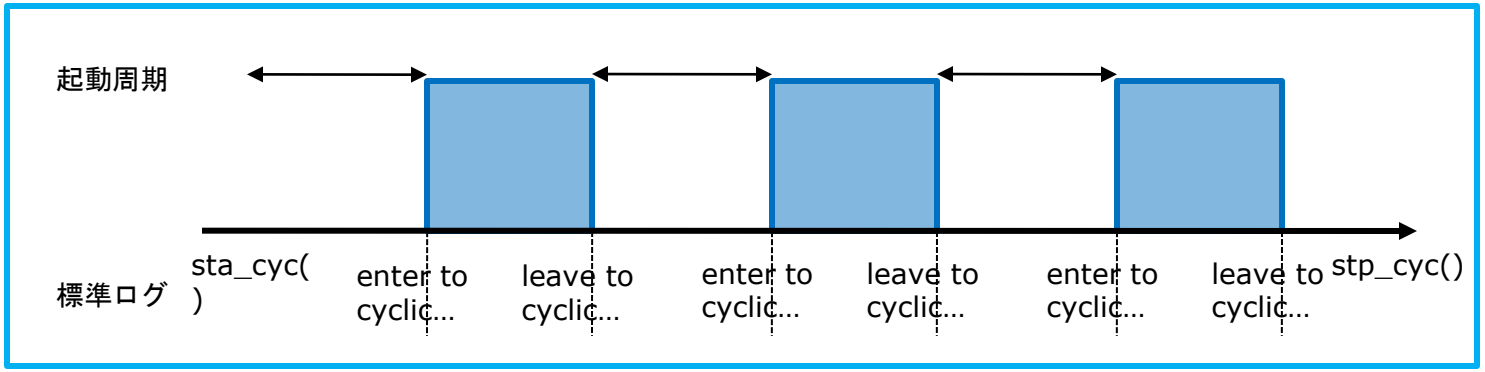
”enter to sta_cyc(or stp_cyc) cycid=%d.”

”leave from sta_cyc(or stp_cyc) ercd=%d.”

”enter to cyclic handler %d.”

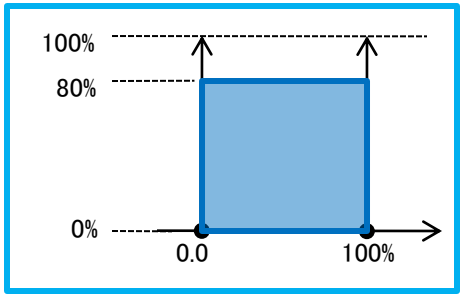
”leave from cyclic handler %d.”

❖標準トレースログだけを対応すると，以下の図になる.



４．周期ハンドラの図形の定義.

❖今回は既存にあるrunningShapesを，背景色だけ変えて定義し，使うようにします.



- Toppers_shapes.vizに定義.
- “handlerDormantShapes”は，dormantShapesを名前だけ変えて定義したもの.

```
“handlerRunningShapes”:[
        {
        "Type":"Rectangle",
        "Size":"100%,80%",
        "Pen":{"Color":"ff0000ff","Width":1},
        "Fill":"6600ff00"
        }
],
"handlerDormantShapes":[
        {
        "Type":"Line",
        "Points":["l(0),0","r(0),0"],
        "Pen":{"Color":"ff999999","Width":1}
        }
]
```

５．周期ハンドラの可視化ルールの定義

❖次は可視化ルールの定義です.

```
"cyclicStateChange":{
        "DisplayName":"状態遷移",
        "Target":"CyclicHandler",
        "Shapes":{
                "callCyclicEvent":{
                        "DisplayName":"状態",
                        "From":"${TARGET}.state",
                        "To":"${TARGET}.state",
                        "Figures":{
                                "${FROM_VAL}==RUNNING":"handlerRunningShapes",
                                "${FROM_VAL}==DORMANT":"handlerDormantShapes"
                        }
                }
        }
}
```

❖ Toppers_rules.vizに定義.

６．出力される図形の確認.

❖希望図形のように出力されたか確認します.