【目次】

1. リリース機能

以下のバージョンに対応した RTE Generator ツールを提供する。

AUTOSAR RTE バージョン	AUTOSAR R3.0 revision002 JASPAR RTE 仕様における下記プロファイルをサポート S0 プロファイル S1 プロファイル
AUTOSAR XML バージョン	・AUTOSAR R3.0 ・AUTOSAR R3.2
RTE バージョン	Specification of RTE Document Version 2.0.0 Final

本ツールでは、JASPAR S0 プロファイル(以下、S0)、および S1 プロファイル（以下、S1）に対応した RTE をサポートする、
リリースツール、およびツールで生成するモジュールの機能概要は、以下のとおり
（詳細は、「６．リリース機能詳細」を参照のこと）

1.1. RTE モジュール

JASPAR RTE で定められている S0／S1 をサポートする。
各プロファイルの概要は以下のとおり

■ S0

ソフトウェアコンポーネント	・シングルインスタンス生成のみサポート ・サポートするコンポーネント種別 - ApplicationSoftwareComponentType, - SensorActuatorComponentType - ComplexDeviceDriverComponentType - EcuAbstractionComponentType
コンポーネント間連携	・Sender-Receiver 連携 - プリミティブ型のデータ連携をサポート （送信、受信、受信タイムアウト、無効化） - ECU 内連携／ECU 間連携をサポート ・Client-Server 連携 - 同期連携（ECU 内連携）をサポート
Runnable Entity 動作	・周期イベント - TimingEvent (OsAlarm,OsScheduleTable で起動) ・Client-Server 連携時に伴うサーバ呼び出しイベント - OperationInvokedEvent ・割り込みの Runnable 起動 API サポート - Rte_Invoke API

S0 プロファイルでは、OS 利用の有無を指定することができる。
・AUTOSAR OS あり（AUTOSAR OS を利用して動作）
・AUTOSAR OS なし（RTE の独自スケジューラ機能 RTE Scheduler を利用して動作）

指定は、ECU コンフィグレーション工程の RTE コンフィグレーション時に行う。

■S1

ソフトウェア コンポーネント	・シングルインスタンス生成のみサポート ・サポートするコンポーネント種別 - ApplicationSoftwareComponentType, - SensorActuatorComponentType - ComplexDeviceDriverComponentType - EcuAbstractionComponentType
コンポーネント間 連携	・Sender-Receiver 連携 - プリミティブ型／コンプレックス型のデータ連携をサポート（送信、送信応答／送信タイムアウト、受信／受信タイムアウト、無効化） - ECU 内連携／ECU 間連携をサポート ・Client-Server 連携 - 同期連携（ECU 内連携）をサポート - 非同期連携（ECU 内連携／ECU 間連携）をサポート
Runnable Entity 動作	・周期イベント - TimingEvent (OsAlarm,OsScheduleTable で起動) ・Sender-Receiver 連携に伴うデータ送信／受信イベント - DataSend/SendCompleted/Receive/ReceivedErrorEvent ・Client-Server 連携時に伴うサーバ呼び出しイベント - OperationInvokedEvent

1.2. RTE Generator ツール

AUTOSAR XML に従って定義された設計データをもとに、RTE モジュールを生成する機能を提供する。

■ サポートする工程
・RTE Contract Phase (コンポーネントのヘッダファイル生成機能)をサポート(S0,S1)
・RTE Generation Phase(RTE モジュール生成機能)をサポート(S0,S1)

2. 前回リリースからの変更履歴

以下では、前回リリースからの主な機能差分について記載する。
詳細な変更履歴は「７．変更履歴詳細」を参照のこと。

2.1. RTE モジュール

■機能追加
１）SW-C 間の内部変数機能対応
InterRunnableVariable 機能をサポート対象機能に追加した。

２）VFB トレーシング対応
VFB Tracing 機能をサポート対象機能に追加した。

2.2. RTE Generator ツール

■機能追加
１）RTE 生成オプションの追加
ツールの起動パラメータを追加し、生成した RTE の内部データを保護するために利用する OS API の選択肢を追加した。

（対応前）：以下の２つのどちらかを選択可能
・SuspendAllInterrupts/ ResumeAllInterrupts
・EnableAllInterrupts /DisableAllInterrupts

（対応後）：以下の３つのうちいずれかを選択可能
・SuspendAllInterrupts/ ResumeAllInterrupts
・EnableAllInterrupts /DisableAllInterrupts
・SuspendOSInterrupts/ ResumeOSInterrupts

２）利用可能な AUTOSAR XML 仕様の拡大
AUTOSAR 3.2 仕様の XML データを読み込んで RTE モジュールを生成可能に対応。
（ただし、RTE モジュールがサポートする機能範囲は、AUTOSAR R3.0 範囲となる）

[注意事項]
- VFB Tracing 機能のサポートに伴い、以前の ECU コンフィグレーションデータを利用した場合、VFB Tracing 機能が有効とものとして RTE モジュール生成が行われる場合がある。
  詳細は、「4.3.10. VFB Tracing 機能に関する注意事項」を参照のこと。

3. 動作環境

以下では、ツールおよびモジュールの動作環境について記載する。

3.1. 必要ソフトウェア

■ RTE Generator ツールの利用
RTE Generator ツールを利用する場合は、以下のソフトウェアが必要になる。

1）Java SE Development Kit 6 以降

2012/xx 月時点では、下記 URL から取得可能。
http://java.sun.com/javase/ja/6/download.html

■ RTE モジュールの利用
RTE Generator ツールで生成した、RTE モジュール（ソースコード）
を利用してアプリケーションを開発／実行する場合は、動作対象マイコンに応じて
以下のソフトウェアが必要になる。
（各開発環境のバージョンは、動作確認を行ったバージョンを記載）

1）V850E 搭載マイコン
ADaC 社製 MULTI4.2.3

動作対象マイコン・エミュレータは、各開発環境／マイコンに応じて準備する必要がある。

3.2. 各マイコン向け開発環境

以下では、各マイコン向け開発環境について、詳細情報を記載する。

○基本的な方針：
　マイコン提供ベンダの推奨オプションを適用

オプション内容の詳細は、各コンパイラの取り扱い説明書を参照のこと

3.2.1. V850E／MUTLI コンパイラ・リンカオプション

Compiler	-G -c -keeptempfiles -noobj -O -OS -OL -sda=all -no_callt
Linker	なし

4. 注意事項

4.1. RTE モジュール利用時の注意事項

4.1.1. RTE の初期化・終了処理

RTE の初期化・終了処理は、以下の RTE API を利用してアプリケーション実装側で行う。

分類	対象	備考
ヘッダファイル	Rte_Main.h	RTE GeneratorツールでRTE生成時に生成
API	Std_ReturnType Rte_Start(void)	RTEの初期化
	Std_ReturnType Rte_Stop(void)	RTEの終了処理

また、RTE の実行制御モジュールの初期化・終了処理は、以下の API を利用してアプリケーション実装側で行う。

分類	対象	備考
ヘッダファイル	Os.h	JASPAR OSから提供 RTE Schedulerを利用する場合は不要
API	/* OSの制御開始 */ void StartOS(AppModeType Mode)	実行制御モジュールの初期化と開始
	/* OSの制御終了*/ void ShutdownOS(StatusType Error)	実行制御モジュールの終了

■RTE の初期化

RTE の初期化は、アプリケーションのスタートアップルーチンから Rte_Start 関数を呼び出して行う。初期化が行われない場合は、RTE API の実行結果は保障されない。

Rte_Start の呼び出しは、実行制御モジュールの開始前（API: StartOS ）の呼び出し前に行う。

例１：アプリケーションの起動処理(JASPAR OS 利用時)

```
#include “Os.h”
/*
 * アプリケーションのスタートアップ処理
 */
int main() {
        :
      /* アプリケーションのデータ領域、利用デバイスの初期化 */
        :

      /* RTE の初期化 */
      Rte_Start();
        :
      /* 実行制御モジュール（AUTOSAR OS or RTE Scheduler）の開始 */
      StartOS(APPMODE);
        :
}
```

例２：アプリケーションの起動処理（RTE Scheduler 利用時）

```
#ifdef USE_RTE_Scheduler
      /* RTE Scheduler 用 API の定義 */
      extern void ShutdownOS(unsigned int);
      extern void Rte_Scheduler_Init(void);
      extern void StartOS(unsigned int);
      /* APPMODE は任意（現仕様では値を利用しない） */
      #define APPMODE 0
#endif

/*
 * アプリケーションのスタートアップ処理
 */
int main() {
        :
      /* アプリケーションのデータ領域、利用デバイスの初期化 */
        :
      /* RTE の初期化 */
      Rte_Start();
      /* RTE Scheduler 初期化 */
      Rte_Scheduler_Init();
      /* 実行制御モジュール（RTE Scheduler）の開始 */
      StartOS(APPMODE);
        :
}
```

■RTE の終了処理
RTE を明示的に停止する場合は、終了処理の割り込みハンドラを実装し、
ハンドラ内から Rte_Stop 関数を呼び出す。

【補足】
AUTOSAR では、ECU State Manager から ECU の停止／開始、および COM、OS の初期化完了通知を受け取り RTE の初期化／終了処理を行う仕様となる。
ECU State Manager なしでシステムを構成する場合は上記方法で初期化する必要がある。

4.1.2. コンポーネントの初期化・終了処理

RTE で動作するコンポーネントの初期化・終了処理は、以下の方法で行う。

アプリケーションのスタートアップルーチンから、C 関数呼び出しなどで独自に
各ソフトウェアコンポーネントの初期化を行う。
このとき、Rte_Start を呼び出して RTE の初期化を完了した後に、各コンポーネントの初期化関数を呼び出す。

また、終了処理に関しても同様にアプリケーション側でシステムの終了タイミングを判定して、各コンポーネントの終了処理を呼び出す。

4.1.3. S0 プロファイル時のスケジューラ（S0）

RTE プロファイル　S0 および S0OS の場合は、AUTOSAR OS の代わりに
RTE 専用の汎用スケジューラ機能「RTE Scheduler」を利用することも可能である。

RTE Scheduler は、AUTOSAR 仕様には含まれない独自モジュール機能であり、
本ツールの RTE 生成時に、生成オプションで –sch を指定した場合にのみ、RTE モジュールの一部として生成される。

4.1.4. モジュールが予約するハードウェアリソース（S0）

S0 で、RTE Scheduler を利用する場合は、RTE の周期動作を実現するため
以下のハードウェアリソースを利用する。

使用リソース	V850 マイコン
タイマ	タイマ A8（TAA9）

4.1.5. COM モジュールとのインタラクション

RTE は、以下のコールバックを利用して AUTOSAR COM からの通知を処理する

分類	COM コールバック <sn/sg>:COM Signal/Group 名	COM からの通知内容
データ送信	Rte_COMCbkTAck_<sn/sg>	データの送信成功
	Rte_COMCbkTErr_<sn/sg>	データの送信失敗（COM エラー）
	Rte_COMCbkTOut_<sn/sg>	データの送信失敗（タイムアウト）
データ受信	Rte_COMCbk_<sn/sg>	データの受信
	Rte_COMCbkInv_<sn>	データの無効化受信
	Rte_COMCbkTOut_<sn>	データの受信失敗（タイムアウト）

【補足】

・データ送信のタイムアウト検出

Rte_COMCbkTOut_<sn/sg>によりデータ送信確認のタイムアウトを検出する。
送信タイムアウトは、AUTOSAR メタモデルで該当要素（例:Sender-Receiver 連携のデータ送信確認であれば、TransmissionAcknowledgementRequest）にタイムアウト時間が設定されていた場合のみ検出する。
タイムアウト時間に 0 が指定された場合は、タイムアウトしないものとみなし上記コールバック関数は生成しない。

・データ受信のタイムアウト検出

Rte_COMCbkTOut_<sn/sg>によりデータ受信間隔のタイムアウトを検出する。
受信タイムアウトは、AUTOSAR メタモデルで該当要素（例:Sender-Receiver 連携のデータ受信であれば、UnqueuedReceiverComSpec）にタイムアウト時間が設定されていた場合のみ検出する。
タイムアウト時間に 0 が指定された場合は、タイムアウトしないものとみなし上記コールバック関数は生成しない。

4.2. コンポーネント実装時の注意事項

4.2.1. RTE API（Implicit Access API）によるデータ更新（S1)

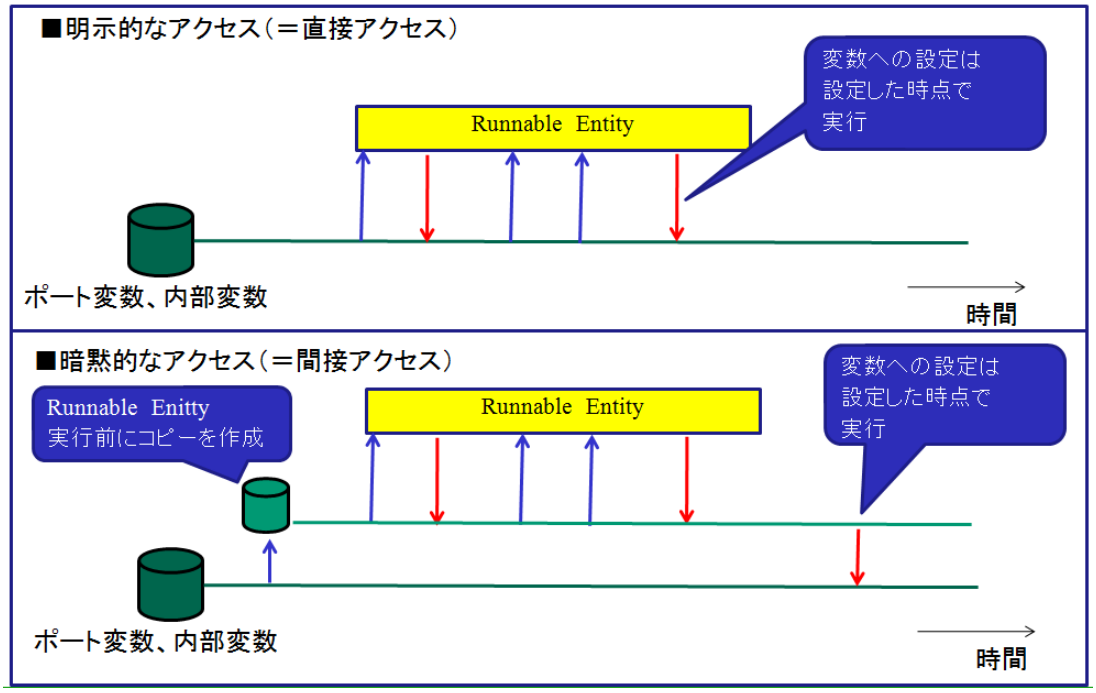
Implicit Write API（間接アクセス API） を利用可能なように設計されている Runnable Entity は、実行時に必ず該当する API を呼び出してデータ値の書き込みを行う必要がある（AUTOSAR 3.0 RTE 仕様）。

【対象 API】

void Rte_IWrite_<re>_<p>_<d> ([IN RTE_Instance], IN <type>)	ポートのデータ値に対してデータバッファを経由した書き込みアクセスを行う
void Rte_IInvalidate_<re>_<p>_<d> ([IN Rte_Instance <instance>])	ポートのデータ値に対してデータバッファを経由した値の無効化を行う

<re>:Runnable Entity 名、<p>:ポート名、<d>:データエレメント名

Implicit Access API は Runnable Entity からアクセスする対象をコピー変数とすることで、変数アクセスのオーバヘッドが発生しないデータ読込／書込を行う。



RTE は、RunnableEntity から実際に Implicit Write API が呼び出されたかどうかにかかわらず、RunnableEntity が終了した後にコピーした変数を実際のデータを表す変数に反映する。
従って、Implicit Write API が呼び出されていない場合は、RunnableEntity の実行後にデータ値が不定となる。

4.2.2. ComplexDeviceDriver, ECU Abstraction のインプリメンテーション

以下では、ComplexDeviceDriver, ECU Abstraction Component を 割り込みコンテキストで動作させる場合のインプリメンテーション方法について記載する。

【注意事項】
ComplexDeviceDriver, ECU Abstraction Component のインプリメントについ
AUTOSAR 仕様では詳細に仕様化されていないため、本ツールで解釈している仕様を記載する。

■要件
AUTOSAR のコンポーネント仕様上、ComplexDeviceDriver, ECU Abstraction は、ECU 依存で高速に動作可能であることが求められる。
従って、これらのコンポーネントの振る舞い（実際の処理コード）は、割り込みコンテキストから実行可能とする必要がある。

■インプリメンテーション
AUTOSAR コンポーネントの振る舞いは、Runnable Entity として定義される。
Runnable Entity は、通常 OS タスクで動作し RTE から自動的に実行される。
ComplexDeviceDriver, ECU Abstraction Component の振る舞いを、割り込みコンテキストから実行するように定義する場合は、以下の手順で定義／実装する。

1. AUTOSAR メタモデルによる振る舞いの定義
AUTOSAR メタモデル仕様では、コンポーネントの振る舞いは、RunnableEntity モデルにより定義される。
割り込みコンテキストで動作する振る舞いを定義する場合は、RunnableEntity を BSW モジュールとして割り込みで動作させることを明示するために、以下のように「BSW-ENTITY-REF」要素を追加してモデル定義を行う。

例：

```
<RUNNABLE-ENTITY>
    <SHORT-NAME>bswRunnable1</SHORT-NAME>
    <!-- BSW Module として割り込みで実行する Runnable Entity 宣言-->
    <BSW-ENTITY-REF DEST="BSW-INTERRUPT-ENTITY"/>
    <!-- BSW Module は、CAN-BE-INVOKE-CONCURRENTLY:true が一般的 -->
```

補足：BSW-ENTITY-REF は、AUTOSAR メタモデル BSWModuleTemplate で定義された BswEntity 要素への参照を設定する。本ツールでは、BswEntity 要素の内容は解析しないため空要素定義で良い。

BSW-ENTITY-REF が定義されていない Runnable Entity は、Application Software Component の Runnable Entity と同様に、RTE によって OS タスクで実行するものとして扱う。

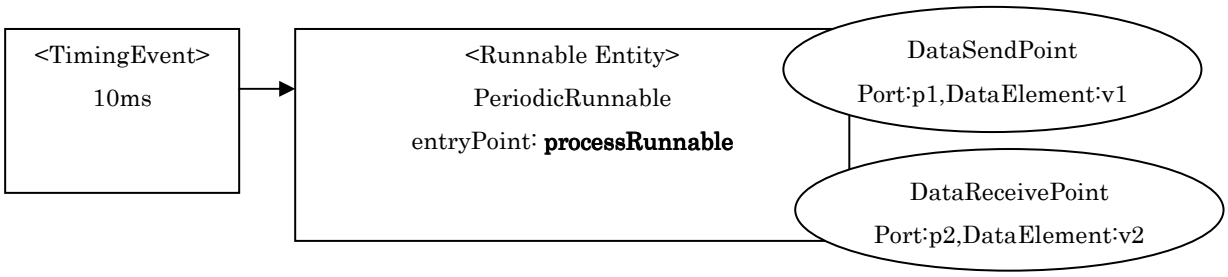
2.振る舞いのインプリメンテーション

○ Runnable Entity のインプリメンテーション

AUTOSAR メタモデル定義で、割り込みコンテキストで動作する Runnable Entity として定義されていた場合、該当する RunnableEntity が利用する RTE API は生成されるが、RTE から Runnable Entity を起動するコードは生成されない。

アプリケーション開発者は、任意の処理タイミング（例：割り込みハンドラ）で呼び出される独自関数を定義し、RTE API を利用して RunnableEntity の処理をインプリメントする。

例：TimingEvent で起動する Runnable Entity
・10ms 周期で起動するように定義した Runnable Entity「PeriodicRunnable」は、Port: p1 に対してデータ書き込みを、Port:p2 からデータ読み込みを行う。



【PeriodicRunnable 用に生成される RTE API】
- Rte_Send/Write_p1_v1 ・・・・・DataSendPoint に対応した書き込み API
- Rte_Read/Receive/Write_p2_v2・・・DataReceivePoint に対応した読み込み API

パターン１：割り込みコンテキストで動作させない場合
Runnable Entity「PeriodicRunnable」の 10ms 周期呼び出し処理は、ECU コンフィグレーションで定義した OS オブジェクトを利用して RTE モジュールにより行われる。

アプリケーション開発者は、生成された RTE API を利用して entryPoint で指定された C 言語の関数本体（例では、processRunnable）を定義することで Runnable Entity「PeriodicRunnable」のインプリメントを行う。

パターン２：:割り込みコンテキストで動作させる場合
10ms 周期のタイマ処理、および Runnable Entity の entryPoint 呼び出し処理は RTE モジュールでは行われない
（他コンポーネントと連携するための RTE API のみ生成される）

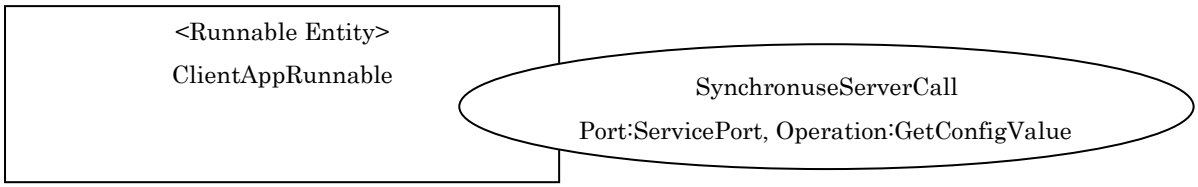
アプリケーション開発者は、10 秒毎に実行されるタイマ割り込みハンドラを独自に定義し、割り込みハンドラから実行される関数内で RTE API を利用して Runnable Entity「PeriodicRunnable」のインプリメントを行う。

○ Server Runnable Entity のインプリメンテーション

OperationInvokedEvent で起動される Runnable Entity（ Server Runnable ） が割り込みコンテキストで動作する Runnable Entity として定義されていた場合、該当する Server Runnable を呼び出す RTE API（Rte_Call API）のプロトタイプ定義のみ生成され RTE API 本体、および Runnable Entity を起動するコードは生成されない。

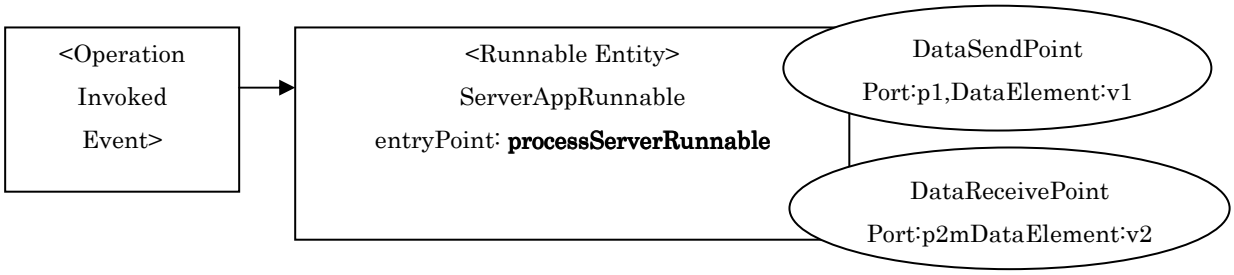
Server Runnable を呼び出す RTE API は、Rte_Call_<ポート名>_<オペレーション名>となる。
例：OperationInvokedEvent で起動する Runnable Entity
- Port「ServicePort」、Operation「GetConfigValue」に対して Client-Server 連携が行われる。
- GetValue オペレーションの呼び出しで、実行される Server Runnable はポートへの書き込み／読み込みを行うものとする。

- Client コンポーネント（ClientApp）



【ClientApp 用に生成される RTE API】
- Rte_Call_ServicePort_ GetConfigValue・・・SynchronuseServerCall に対応した Server 呼び出し RTE API
  - 引数は、Client-Server 連携のインターフェースに依存
  - 割り込みコンテキストで動作させるかどうかで生成される範囲が異なる
    - 割り込み動作ではない：RTE API の本体を RTE が生成
    - 割り込み動作である　：RTE API の関数プロトタイプ宣言のみ生成

- Server コンポーネント（ServerApp）
  Rte_Call API 呼び出しに伴い実行される Runnable Entity 定義



【ServerAppRunnable 用に生成される RTE API】
- Rte_Send/Write_p1_v1 ・・・・・DataSendPoint に対応した書き込み API
- Rte_Read/Receive/Write_p2_v2・・・DataReceivePoint に対応した読み込み API

パターン１：割り込みコンテキストで動作させない場合
Runnable Entity「ServerAppRunnable」の呼び出し処理は、ECU コンフィグレーションの定義に従って RTE モジュールにより行われる。

アプリケーション開発者は、ServerAppRunnable 用に生成された RTE API を利用して entryPoint で指定された C 言語の関数本体（例では、processServerRunnable）を定義することで Runnable Entity 「ServerAppRunnable」のインプリメントを行う。

パターン２：:割り込みコンテキストで動作させる場合
Runnable Entity の entryPoint 呼び出し処理は RTE モジュールでは行われない

アプリケーション開発者は、Rte_Call API のインプリメンテーションとなる以下の関数本体を定義することで Server Runnable の処理をインプリメントする。
Rte_Call_<コンポーネントタイプ名>_<ポート名>_<オペレーション名>
※：RTE API シンボルの競合を避けるため、シングルインスタンスの RTE API 関数命名則に従って、コンポーネントタイプ名を付加する。

例では、Rte_Call_ClientApp_ServicePort_ GetConfigValue 関数を定義し、ServerAppRunnable 用に生成された RTE API を利用して Runnable Entity「ServerAppRunnable」のインプリメントを行う。

■割り込みコンテキストでの動作制限
割り込みコンテキストから RTE API を実行する場合は、以下の制限事項がある。

１）Implicit access API は利用できない

２）カテゴリ１ISR から RTE API を実行した場合、データの反映のみ行われる。
OS タスクの起動／再開を伴う RTE イベント処理は行われない。
例：Sender-Receiver 連携
割り込みコンテキストから Rte_Write API を実行した場合、データ値のみ反映され、データ受信イベントで動作する RunnableEntity は起動されない。

理由：AUTOSAR OS ではカテゴリ１ISR からのシステムコールがサポートされないため

例のように、RTE API の実行に伴い OS タスクコールが発生する場合は、カテゴリ２ISR を利用して Runnable 処理を実装する必要がある。

4.3. ECU コンフィグレーションに関する注意事項

4.3.1. Runnable Entity の割り込み実行定義

Runnable Entity の動作 OS タスクは、「Rte Configuration Editor」ツールを利用して「RunnableEntityMapping」項目を設定することにより指定する。
「RunnableEntityMapping」項目では、Runnable Entity の起動トリガとなる RTE イベントと Runnable Entity が動作するコンテキストを表すオブジェクトを設定する。

【Rte Configuration 項目（抜粋）】

項目	設定内容
SwComponentInstance	
+ RunnableEntityMapping	
+ MappedToTaskRef	RTE イベントで起動する Runnable Entity が動作するコンテキスト A）OS タスク B）ISR
+ RTEEventRef	Runnable Entity の起動トリガとなる RTE イベント

MappedToTaskRef 項目に、OS タスクを割り当てた場合は、AUTOSAR 標準仕様に準拠した動作となり、割り当てた OS タスクで動作する。
Runnable Entity は RTE イベントの定義内容に従って RTE から自動的に実行される。

MappedToTaskRef 項目に、OS の ISR を割り当てた場合は、Runnable Entity を実行する以下の API が生成される。

void Rte_Invoke_<コンポーネントタイプ名＞_<Runnable Entity 名>()

この設定を行った場合は、RTE イベントの定義は無視される。
代わりに、コンポーネント開発者が、任意の割り込み処理を定義して上記 API を呼び出すことで RunnableEntity を動作させることができる。

4.3.2. TimingEvent を実現するタイマの設定 (S0)

RTE イベント「TimingEvent」は、周期的に実行されることを表すイベントである。
S0 の ECU コンフィグレーションでは実行周期を実現するタイマは、AUTOSAR OS の OsAlarm オブジェクトで定義する。
S0 では、タイマを必ず自動起動するため以下の項目を設定する必要がある。

【Os Configuration 項目（抜粋）】

項目	設定内容
OsAlarm	
+ OsAlarmAutostart	S0 プロファイル用の設定では必ず自動起動
+ OsAlarmAlarmTime	周期処理開始までの遅延時間（単位: ms ）
+ OsAlarmCycleTime	周期実行間隔（単位: ms）

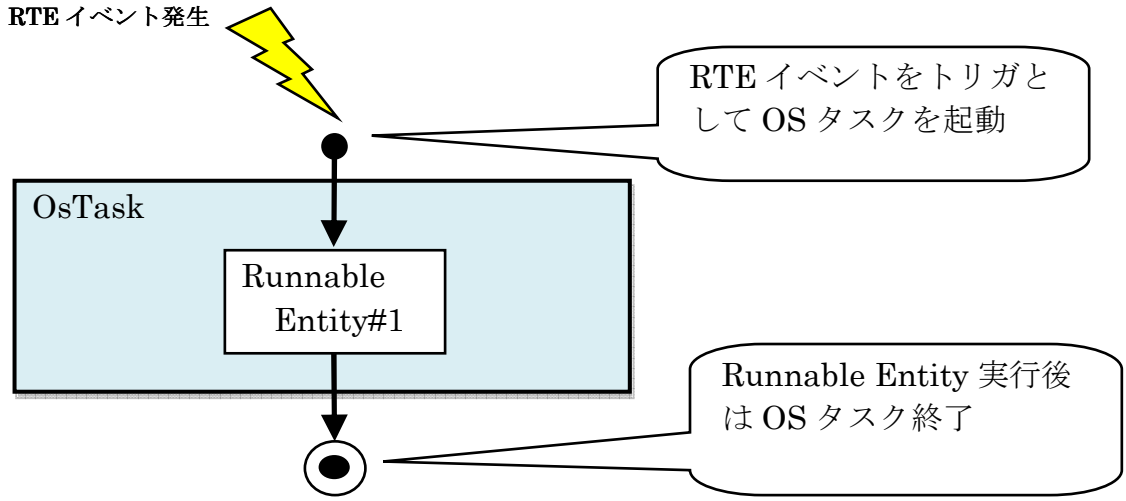
4.3.3. Runnable Entity の動作設定 (S1)

Rte Configuration Editor を利用して、以下の Runnable Entity 動作設定を行うことができる。

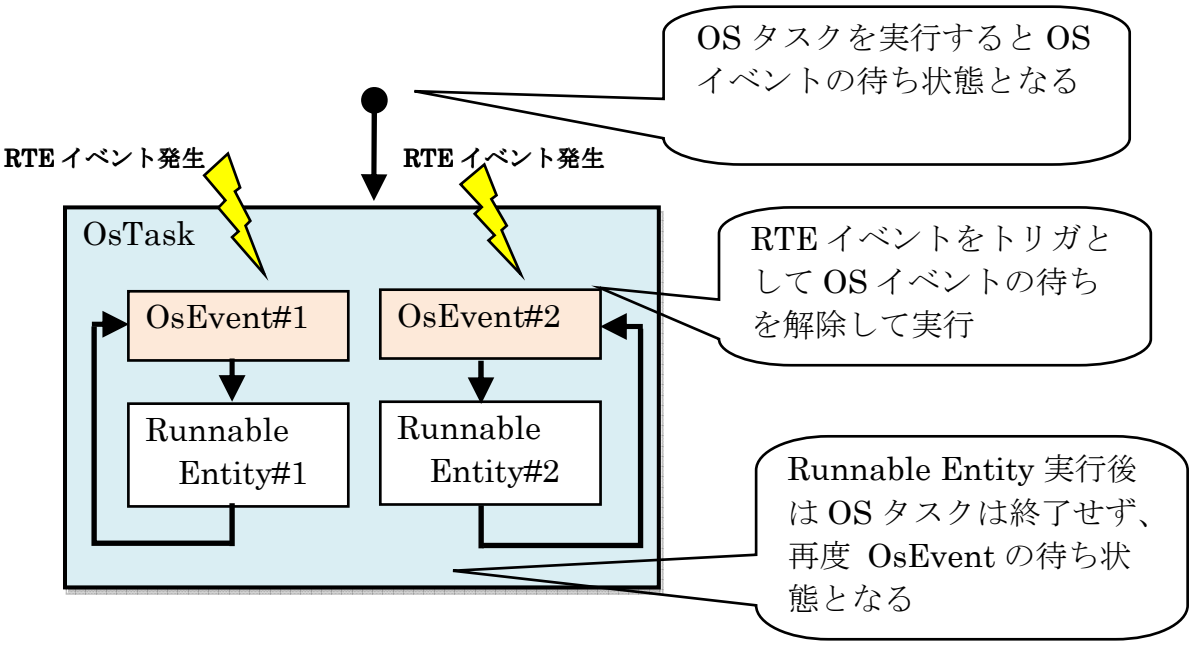
A) 動作する OS タスクを指定する
B) 動作する OS タスクと、動作トリガに利用する OS イベントを指定する
C) 動作する OS タスクを指定しない（同期 Client-Server 連携の ServerRunnable のみ）
D) 動作する ISR を指定する

動作する OS タスクが指定された場合、RTE は以下のような Runnable Entity 制御を行う。

A) 動作する OS タスクが指定された場合



B) 動作する OS タスク、および OS イベントが指定された場合



一般的に、常時動作する B）のほうが Runnable Entity は高速に動作する。
また、１つの OS タスクに、異なる RTE イベントで動作する Runnable Entity を割りつけて OS タスクを共有できるため消費資源量を低減することができる。
反面、ほぼ同時に異なる RTE イベントが発生する場合、先に発生した RTE イベントで動作する Runnable Entity の処理が完了するまでは、後から発生した RTE イベントが処理されないため、実行タイミングに留意する必要がある。

■ Rte コンフィグレーション設定時の注意事項

OS イベントを利用して Runnable Entity を実行するように設定する場合は、OS タスクを拡張タスクに設定し、OS 開始時に自動起動するように設定する必要がある。

【設定内容】
OS タスクに OS イベントを割り当てて、拡張タスクとして扱うように設定する。
また、システムのスタートアップ後に OS タスクを実行してイベント待ち状態になるように、OS タスクのオートスタート設定を有効にする。

【Os Configuration 項目（抜粋）】

項目	設定内容
Task	
+ OsTaskEventRef	該当タスクから利用する OS イベント
+ OsTaskAutostart	OS 開始時にオートスタートを行うかどうかの設定
+ OsTaskAppModeRef	タスクを開始するアプリケーションモードの指定

4.3.4. RTE API を呼び出す Runnable Entity の OS タスク設定 (S1)

Runnable Entity が動作した場合に実行される関数（Runnable Entity のエントリポイント関数）の実装コード内で、ポートのデータ値を更新する API、および サーバ呼び出し API を呼び出した場合は、コンポーネント間の連携が行われる。

このとき、データの受信イベントや、サーバ呼び出しイベントが定義されている場合は、イベントをトリガとして Runnable Entity が動作し、その動作タスクは、Rte Configuration Editor で定義された OS タスクとなる。

■ Rte コンフィグレーション設定時の注意事項
ECU 内連携の場合は、同一 OS 上での動作となる。
API 呼出しにより起動される Runnable Entity の OS タスクの優先順位が低い場合は、API を呼び出した RunnableEntity が動作する OS タスクから複数の OS タスクが呼び出されることになる。

API を呼び出した RunnableEntity の動作する OS タスクが、他の Runnable Entity（の OS タスク）を起動できるようにするため「Rte・Os Configuration Editor」ツールによるコンフィグレーションで以下の設定を行う。

【設定内容】
RTE API を呼び出す Runnable Entity に割り当てた OS タスクに対して
動作する可能性のある OS タスク数を「OsTaskActivation」項目に設定する

【Os Configuration 項目（抜粋）】

項目	設定内容
Task	
＋ OsTaskActivation	該当タスクから実行することが可能な最大タスク数

4.3.5. Runnable Entity のクリティカルセクション設定 (S1)

○排他エリアを利用したクリティカルセクションの定義
AUTOSAR メタモデルでは、ExclusiveArea を利用したクリティカルセクションの作成方法として、以下の２種類を定義することができる。

[a]. Runs In アクセス
RTE がクリティカルセクションを生成する。保護される区間は、Runnable Entity が呼び出される直前から、終了するまでとなる。
[b]. Can Enter アクセス
RunnableEntity の実装コード内で任意の区間を保護する。保護される区間は、
Rte_Enter API を呼び出してから、 Rte_Exit API を呼び出すまでの間となる。

○排他エリアのコンフィグレーション
排他エリアは「Rte Configuration Editor」ツールで実現方法を指定することができる。
【Rte Configuration - ExclusiveAreaImplMechanism 項目（抜粋）】

Cooperative Runnable Placement	利用不可能（機能制限）
Interrupt Blocking	OS を割込禁止にしてクリティカルセクションを作成
Non Preemptive Tasks	処理なし（OS のタスク設定で NonPreempt に設定しておく）
Os Resource	利用不可能（機能制限）

■排他エリアのアクセス、および実現メカニズム設定時の注意事項
クリティカルセクションが「割込禁止」で実現されている場合、クリティカルセクションの保護区間内から、RTE API（※）を呼び出した場合に、受信イベントなどの RTE イベントをトリガとして動作する Runnable Entity は実行されない。
これは、AUTOSAR OS 仕様で、割込禁止区間から OS タスクを起動できないことに起因する。

※間接アクセスを行う Implicit Write API を利用する場合は、RunnableEntity の実行は保護区間外となるため動作することが保障される。

割り込み禁止による保護により RunnableEntity が起動されない場合は、RTE API を呼び出すコンポーネント実装側で、以下の対応を行う必要がある。

a.クリティカルセクションの保護区間を変更する
「Runs In アクセス」を「Can Enter アクセス」に変更して、RTE API が保護区間内から実行されないように実装を修正する

b.排他エリアの実現方法を変更する
「Interrupt Blocking」以外の実現メカニズムを適用する（例：OS リソース）

4.3.6. 同期 Client-Sever 連携の動作設定

同期 Client-Server 連携は、同一 ECU 内の連携のみサポートする。
Server の Runnable Entity は、Client の Runnable Entity と同じ実行コンテキストでの動作のみをサポートする。

■ Rte コンフィグレーション設定時の注意事項

Server Runnable Entity の動作コンテキストは、「Rte Configuration Editor」ツールを利用して「RunnableEntityMapping」項目を設定することにより指定する。

「RunnableEntityMapping」項目では、Runnable Entity の起動トリガとなる RTE イベントと Runnable Entity が動作する OS タスクを設定する。

【Rte Configuration 項目（抜粋）】

項目	設定内容
SwComponentInstance	
+ RunnableEntityMapping	
+ MappedToTaskRef	RTE イベントで起動する Runnable Entity が動作する OS タスク
+ RTEEventRef	Runnable Entity の起動トリガとなる RTE イベント

AUTOSAR 仕様では、「MappedToTaskRef」項目を定義しない、又は項目値を空文字列とすることで、呼び出し元の動作コンテキストでの動作指定となる。
Client-Server 連携を、関数呼出しと同様に動作させる場合は、呼出し元の動作コンテキストで動作するように設定する。

4.3.7. 非同期 Client-Sever 連携の動作設定 (S1)

非同期 Client-Server 連携は、ECU 内／ECU 間の連携をサポートする。

Server の Runnable Entity 、および Server の応答で動作する RunnableEntity は、OS タスクでの動作のみサポートする。

■ **Rte コンフィグレーション設定時の注意事項**

Server Runnable Entity の動作コンテキストは、「Rte Configuration Editor」ツールを利用して「RunnableEntityMapping」項目を設定することにより指定する。

「RunnableEntityMapping」項目では、Runnable Entity の起動トリガとなる RTE イベントと Runnable Entity が動作する OS タスクを設定する。

【Rte Configuration 項目（抜粋）】

項目	設定内容
SwComponentInstance	
+ RunnableEntityMapping	
+ MappedToTaskRef	RTE イベントで起動する Runnable Entity が動作する OS タスク
+ RTEEventRef	Runnable Entity の起動トリガとなる RTE イベント

非同期 Client-Server 連携は、非同期で要求送信と応答受信を行うことから、必ず OS タスクを設定する必要がある。

4.3.8. TimingEvent を実現する OS オブジェクトの設定 (S1)

RTE イベント「TimingEvent」は、周期的に実行されることを表すイベントである。
コンフィグレーションでは、実行周期を実現するための Os Alarm / Os Schedule Table を「Os Configuration Editor」で設定する。

本バージョンの RTE Generator ツールでは、以下のルールに従って「TimingEvent」に対応する OsAlarm / OsScheduleTable の設定を行う必要がある。

１）Os Alarm / Os Schedule Table は、対応する TimingEvent で動作する Runnable Entity の OS タスクを、TimingEvent の周期通りに呼び出すように設定する

２）Os Alarm / Os Schedule Table は、OS 起動時のオートスタート設定を有効にし、OS を開始すると自動的に開始されるよう設定する。

３）Os Alarm / Os Schedule Table の OS タスク呼出し周期は、基本的に TimingEvent の周期通りに設定する。

[補足] OsAlarm を共有した TimingEvent 処理の実現

異なる周期の TimingEvent に対して、同じ OS オブジェクトを割り当てる場合は、

OsAlarm を利用して定義する。
この場合、対応するそれぞれの TimingEvent の周期の最大公約数となる周期で
OS タスクが実行されるように設定する。

例：TimingEvent の内容に応じた OsAlarm の設定内容

番号	コンフィグレーション内容	OsAlarm の設定内容
例 1	100ms 周期で起動するイベント「Timing100ms」に対し、OsAlarm「AppAlarm」を割り当てる場合	AppAlarm の起動周期：100ms
例 2	100ms 周期で起動するイベント「Timing100ms」、「Timing01s」 に対して OsAlarm「AppSharedAlarm」を割り当てる場合	AppSharedAlarm の起動周期：100ms
例 3	100ms 周期で起動するイベント「Timing100ms」, 30ms 周期で起動するイベント「Timing30ms」 に対して、OsAlarm「AppSharedAlarm」を割り当てる場合	AppSharedAlarm の起動周期：10ms

4）TimingEvent で起動する RunnableEntity に対し、Rte コンフィグレーションで ActivationOffset （実行周期のオフセット指定） が設定されている場合は、
対応する Os Alarm / Os Schedule Table の初回実行までのオフセット時間に反映する。

■ Rte・Os コンフィグレーション設定時の注意事項

Runnable Entity の動作 OS タスクは、「Rte Configuration Editor」ツール
を利用して「RunnableEntityMapping」項目を設定することにより指定する。
「RunnableEntityMapping」項目では、Runnable Entity の起動トリガとなる RTE
イベントと Runnable Entity が動作する OS タスクを設定する。

【Rte Configuration 項目（抜粋）】

項目	設定内容
SwComponentInstance	
+ RunnableEntityMapping	
+ MappedToTaskRef	RTE イベントで起動する Runnable Entity が動作する OS タスク
+ RTEEventRef	Runnable Entity の起動トリガとなる RTE イベント

周期実行処理を実現する OS オブジェクトは、「Os Configuration Editor」ツールを利用して、オブジェクト毎に自動起動タスクを設定することができる。

【Os Configuration 項目（抜粋）】

項目	設定内容
OsScheduleTableActivateTaskRef	Os Schedule Table から自動的に実行する OS タスク
OsAlarmActivateTaskRef	Os Alarm から自動的に実行する OS タスク

・「SwComponentInstance」-「RunnableEntityMapping」-「RTEEventRef」項目の対象が TimingEvent の場合は、「SwComponentInstance」-「RunnableEntityMapping」-「MappedToTaskRef」で指定した OS タスクに、以下のどちらかの項目を指定して、OS オブジェクトから自動的に起動される対象 OS タスクに設定する必要がある。

・「OsAlarm」項目の「OsAlarmActivateTaskRef」項目
・「OsScheduleTable」 項目の「OsScheduleTableActivateTaskRef 」項目

・異なる周期の TimingEvent に対して同じ OsAlarm を割り当てる場合は、RTE 側で OsAlarm の実行周期を認識するために、以下の設定を行う必要がある。
対象：OsAlarm で利用する OsCounter
設定項目：OsCounter - OsSecondsPerTick

【Os Configuration 項目（抜粋）】

項目	設定内容
OsCounter	OsAlarm で利用するカウンタの定義
OsSecondsPerTick	OsCounter のカウント値（TickCount）当たりの時間(s)

OsSecondsPerTick 項目で、OsAlarm の実行周期が定義されていない場合は、
1 ms / 1 TickCount として扱う

4.3.9. 排他エリアの実現方法に関する注意事項 (S1)

○排他エリアのコンフィグレーション
コンポーネントに定義された排他エリアは、「Rte Configuration Editor」を利用して以下の項目により実現方法を指定することができる。

【Rte Configuration - ExclusiveAreaImplMechanism 項目（抜粋）】

Cooperative Runnable Placement	OS リソースを利用してクリティカルセクションを作成
Interrupt Blocking	OS を割込禁止にしてクリティカルセクションを作成
Non Preemptive Tasks	処理なし（OS のタスク設定）
Os Resource	OS リソースを利用してクリティカルセクションを作成

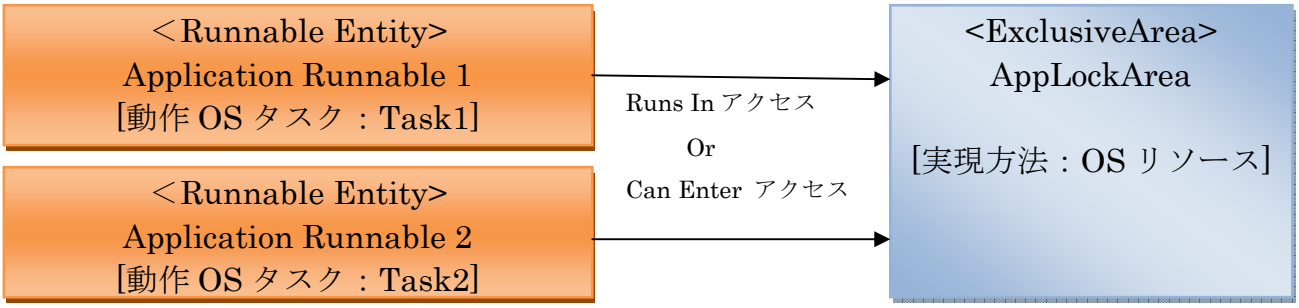
このとき、実現メカニズムとして、「Cooperative Runnable Placement」「Os Resource」が指定された場合は、OS リソースを利用してクリティカルセクションを実現する。

排他エリアの実現メカニズムに OS リソースを利用するオプションが指定された場合は、「**ExclusiveArea 要素と同じ名称の OS リソース**」を利用してクリティカルセクションの実装に利用する。
OS コンフィグレーションで、該当する OS リソースを定義する必要がある。

■排他エリアの実現メカニズム設定時の注意事項

○OS リソースの割り当て（ECU コンフィグレーション）

例：排他エリアのアクセス例



このような場合、OS リソース「AppLockArea」を定義し、OS タスク「Task1」と「Task2」は OS リソース「AppLockArea」を利用可能なように定義する必要がある。

OS タスク、OS リソースの設定は「Os Configuration Editor」ツールを利用して行う。
OS タスクで利用する OS リソースは、以下の項目で設定することができる。

【Os Configuration 項目（抜粋）】

項目	設定内容
OsTaskResourceRef	OS タスクから利用可能な OS リソース

なお、条件に該当する OS リソースが発見できない場合、RTE はクリティカルセクションを OS リソースではなく、割込禁止により実現する。

○RunsIn アクセス利用時のロック実現メカニズム（ECU コンフィグレーション）
Runnable Entity が RunsIn アクセスを行うように設定されていた場合、RTE は該当する排他エリアのロックを取得した状態で、該当する Runnable Entity を呼び出す。

アクセス対象となる排他エリアが割込禁止で実現される場合、Runnable Entity 実行中は常に割込禁止の状態となるため、RTE API を呼び出した結果発生するイベント処理（OS タスク起動）を実現する OS　API 呼出し動作が保障されなくなる。
RunIn アクセスで排他エリアを利用する場合は、OS リソースなど割込禁止以外のメカニズムを利用して、排他エリアを実現するようにコンフィグレーションする必要がある。

4.3.10. VFB Tracing 機能に関する注意事項

Version1.1.0 より、VFB Tracing 機能をサポートしている。
Version1.1.0 以前のバージョンで利用していた ECU コンフィグレーションデータを利用する場合、以下の非互換が発生する。

○条件
データ内の RTE コンフィグレーション設定において、VBF Tracing 項目設定を行っていない場合、または有効となるように設定されていた場合

○影響
本ツールで生成した RTE モジュールが、VFB Tracing 機能が全て有効な実装コードとして生成される。

[回避方法]
Version1.1.0 以前のバージョンで利用していた ECU コンフィグレーションデータを利用する場合は、RTE モジュールのコンフィグレーションに対して、VFB Tracing 機能を OFF にするよう設定を行う。
■VFB Tracing 機能を OFF にする場合の設定例（下記赤字範囲）

```
<CONTAINER>
    <SHORT-NAME>RteGeneration</SHORT-NAME>
    <DEFINITION-REF DEST="PARAM-CONF-CONTAINER-DEF">/AUTOSAR/Rte/RteGeneration</DEFINITION-REF>
                                                    : （省略）
    <PARAMETER-VALUES>
        <ENUMERATION-VALUE>
            <DEFINITION-REF
DEST="ENUMERATION-PARAM-DEF">/AUTOSAR/Rte/RteGeneration/RteOptimizationMode</DEFINITION-REF>
            <VALUE>RUNTIME</VALUE>
        </ENUMERATION-VALUE>
        <INTEGER-VALUE>
            <DEFINITION-REF
DEST="INTEGER-PARAM-DEF">/AUTOSAR/Rte/RteGeneration/RteVfbTrace</DEFINITION-REF>
            <VALUE>0</VALUE>
        </INTEGER-VALUE>
        <FUNCTION-NAME-VALUE>
            <DEFINITION-REF
DEST="FUNCTION-NAME-DEF">/AUTOSAR/Rte/RteGeneration/RteVfbTraceFunction</DEFINITION-REF>
        </FUNCTION-NAME-VALUE>
    </PARAMETER-VALUES>
                                                    : （省略）
```

5. 未実装機能・制限事項

以下では、RTE モジュール／RTE Generator ツールの未実装機能、制限事項について記載する（「６．リリース機能詳細」で「JASPAR プロファイルの仕様範囲外」としている機能は提供対象外であるため記載しない）。

5.1. RTE モジュール

【未実装機能】
・なし

5.2. RTE Generator ツール

【未実装機能】
・以下のコンフィグレーション設定の反映は未実装
　　－最適化オプション指定（実行速度、メモリ効率）

【制限事項】
・なし

6. リリース機能詳細

以下では、本ツールで生成可能な RTE モジュールの機能仕様範囲について説明する。

【表記方法】

表記シンボル	シンボルの意味
●	AUTOSAR RTE 仕様に準拠する。
▲	部分制約。AUTOSAR RTE 仕様に対して制約あり
×	未サポート
―	未サポート（JASPAR プロファイル機能に含まれない）

6.1. RTE 機能仕様

6.1.1. ソフトウェアコンポーネントのインスタンス

ID	分類	機能	RTE API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
AC1	RTE and AUTOSAR Software-Components	ソフトウェアコンポーネント種別		4.1.2.1	●	●
		CompositionType		4.1.2.1	●	●
		CalprmComponentType		4.1.2.1	–	–
		ApplicationSoftwareComponentType		4.1.2.1	●	●
		SensorActuatorComponentType		4.1.2.1	●	●
		ServiceComponentType		4.1.2.1	–	–
		EcuAbstractionComponentType		4.1.2.1	●	●
		ComplexDeviceDriverComponentType		4.1.2.1	●	●
		ポート、インタフェース、コネクタ		4.1.2.2	●	●
		Sender-Receiver Interface		4.1.2.2	●	●
		Client-Server Interface		4.1.2.2	●	●
		Calprm Interface		4.1.2.2	–	–
		Assembly Connector		4.1.2.2	●	●
		Delegation Connector		4.1.2.2	●	●
		Service Connector		4.1.2.2	–	–
		コンポーネント動作（ Internal Behavior）		4.1.2.3	●	●
		インプリメンテーション		4.1.2.4	–	–
AC2	Instantiation	シングルインスタンス生成		4.1.3.3	●	●
		マルチインスタンス生成		4.1.3.4	–	–

		Per Instance Memory		4.1.3.4.3 5.2.5 5.6.1.2	–	–
AC3	RTE and AUTOSAR Service	Standardized AUTOSAR Interface 連携		4.1.4	–	–
AC4	RTE and ECU Abstraction	AUTOSAR Interface 連携 （BswEntity は RTE 制御対象外）		4.1.5	●	●
AC5	RTE and Complex Device Driver	AUTOSAR Interface 連携 （BswEntity は RTE 制御対象外）		4.1.6	●	●

6.1.2. Runnable とデーター貫性

ID	分類	機能	RTE API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	S1
OS						
RE1	Runnable Entity	Runnable の動作カテゴリ		4.2.2.2	●	●
		カテゴリ 1a		4.2.2.2	–	●
		カテゴリ 1b		4.2.2.2	●	●
		カテゴリ 2		4.2.2.2	–	–
RE2	RTE Events	イベントによる Runnable 起動		4.2.2.3	●	●
		TimingEvent		4.2.2.3	●	●
		DataReceivedEvent		4.2.2.3	–	●
		DataReceiveErrorEvent		4.2.3.3	–	●
		DataSendCompletedEvent		4.2.3.3	–	●
		OperationInvokedEvent		4.2.3.3	●	●
		AsynchronousServerReturnsEvent		4.2.3.3	–	●
		ModeSwitchEvent		4.2.3.3	–	–
RE3	RTE Events	Blocking Wait 解除による Runnable 再開		4.2.3.3	–	–
		DataReceivedEvent	Rte_Receive	4.2.3.3	–	–
		DataSendCompletedEvent	Rte_Feedback	4.2.3.3	–	–
		AsynchronousServerReturnsEvent	Rte_Result	4.2.3.3	–	–
RE4	Mapping of runnable entities to tasks	OS 基本タスクの Runnable マッピング		4.2.2.4	–	●
		OS 拡張タスクの Runnable マッピング		4.2.2.4	–	●
RE5	Activation Offset for runnable	Runnable アクティベーションオフセット指定 （TimingEvent のみ）		4.2.2.5	●	●
RE6	Activation and Start of Runnable Entities	Runnable の最低実行間隔定義 （TiminigEvent を除く）		4.2.2.6	–	–
Interrupt decoupling and notifications						
RE6	Basic notification principles	Standardized interface に通知ハンドリング （COM、OS からのコールバック受信）		4.2.3.1	●	●
		AUTOSAR interface による通知ハンドリング		4.2.3.1	●	●

RE7	Decoupling interrupts on RTE level	OS タスクでの Runnable 動作		4.2.3.3	●	●
		割り込みでの Runnable 動作防止		4.2.3.3	●	●
Data Consistency						
DC1	Exclusive Areas	Runnable の動作制御		4.2.4.5	●	●
		RTE による Runnable 実行の排他制御 (Runs-In Exclusive Area)		4.2.4.5	●	●
		RTE API による排他制御 (Can-Use ExclusiveArea)	Rte_Enter Rte_Exit	4.2.4.5	●	●
DC2	Assignment of data consistency mechanisms	Exclusive Area の実現メカニズム設定		4.2.4.5.1	●	●
		Interrupt Blocking		4.2.4.5.1	●	●
		OSResource		4.2.4.5.1	–	●
		NonPreemptive Task		4.2.4.5.1	●	●
		Cooperative Runnable Placement		4.2.4.5.1	–	●
DC3	InterRunnableVariables	InterRunnableVariables の間接アクセス	Rte_IrvIRead Rte_IrvIWrite	4.2.4.6.1	●	●
		InterRunnableVariables の直接アクセス	Rte_IrvRead Rte_IrvWrite	4.2.4.6.2	●	●
Multiple trigger of Runnables						
DC4	Concurrent activation	Runnable の並行実行サポート （Client-Server の同期呼び出し）		4.2.5	●	●
DC5	Activation by several RTEEvents	複数の RTE イベントからの Runnable アクティベーション		4.2.5	●	●

6.1.3. 測定と較正

ID	分類	機能	RTE API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	S1
Measurement						
MS1	What can be measured	Port 間連携の計測		4.2.6.2	–	–
		Sender-receiver 連携のデータ計測		4.2.6.2.2	–	–
		Client-Server 連携のデータ計測		4.2.6.2.2	–	–
MS2	What can be measured	InterRunnableVariables のデータ計測		4.2.6.2	–	–
MS3	RTE support for Measurement	未接続／互換ポートのサポート		4.2.6.2.2	–	–
MS4	RTE support for Measurement	測定無効オプションのサポート		4.2.6.2.2	–	–
Calibration						
CB1	Calibration parameters	キャリブレーションパラメータ定義		4.2.6.3.1	–	–
		共通パラメータ (defined in CalprmComponentType)	Rte_CalPrm	4.2.6.3.1	–	–

		コンポーネントタイプ用パラメータ (defined in SW-Cs)	Rte_CData	4.2.6.3.1	–	–
		コンポーネントインスタンス用パラメータ (defined in SW-Cs)	Rte_CData	4.2.6.3.1	–	–
		キャリブレーションパラメータの分離		4.2.6.3.1.1	–	–
CB2	Support for offline calibration	キャリブレーション無効オプション のサポート		4.2.6.3.2	–	–
CB3	Support for online calibration	Data emulation without SW support (direct access)		4.2.6.3.4	–	–
	Support for online calibration	Data emulation with SW support		4.2.6.3.5	–	–
		Single pointered method		4.2.6.3.5.1	–	–
		Double pointered method		4.2.6.3.5.2	–	–
		InitRAM parameter method		4.2.6.3.5.3	–	–

6.1.4. Sender-Receiver 連携

ID	分類	機能	RTE API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	S1
Sender-Receiver						
SR1	Introduction	ＥＣＵ内のデータ連携		4.3.1.1	●	●
		ＥＣＵ間のデータ連携（COM を利用）		4.3.1.1	●	●
SR2	Receive Modes	データ値の取得 (間接データリードアクセス)	Rte_IRead Rte_IStatus	4.3.1.2	–	●
		データ値の取得 (直接データリードアクセス)	Rte_Read Rte_Receive	4.3.1.2	●	●
		受信イベントからの Runnable 起動 (events to wake up runnable entities)		4.3.1.2	–	●
		受信イベントからの Runnable 起動 (activation of runnable entity)		4.3.1.2	–	●
SR3	Multiple Data Elements	データエレメント定義		4.3.1.3	●	●
		1 データエレメント／ポート のサポート		4.3.1.3	●	●
		n データエレメント／ポート のサポート		4.3.1.3	●	●
SR4	Initial Values	Last is best semantics		4.3.1.3.1	●	●
		Queue semantics		4.3.1.3.1	—	—
SR4	Multiple Receivers and Senders	Sender-Receiver 連携パターン		4.3.1.4	●	●
		sender 1: receiver 1 の連携		4.3.1.4	●	●
		sender 1: receiver n の連携		4.3.1.4	●	●
		sender m : receiver 1 の連携		4.3.1.4	●	●
SR5	Implicit Data Reception and Transmission	間接データ送信／受信		4.3.1.5.1	–	●

		データの間接書き込み （runnable 処理後にデータ反映）	Rte_IWrite	4.3.1.5.1	－	●
		データの間接読み込み （runnable 処理前のデータをコピー）	Rte_IRead Rte_IStatus	4.3.1.5.1	－	●
SR6	Explicit Data Reception and Transmission	直接データ送信／受信		4.1.3.5.2	●	●
		データの直接書き込み(non-blocking) （即時にデータ反映）	Rte_Send Rte_Write	4.1.3.5.2	●	●
		データの直接書き込み(blocking) （書き込み後 ACK 応答が戻るまで wait）	Rte_Send Rte_Write Rte_Feedback	4.1.3.5.2	－	－
		データの直接読み込み（non-blocking）	Rte_Read Rte_Receive	4.1.3.5.2	●	●
		データの直接読み込み（blocking） （WaitPoint による runnable の再開）	Rte_Receive	4.1.3.5.2	－	－
SR7	Transmission Acknowledgement	送信成功応答（ACK 応答）		4.3.1.6	－	●
		応答ステータスの取得（non-blocking）	Rte_Feedback	4.3.1.6	－	●
		応答ステータスの取得（bocking）	Rte_Feedback	4.3.1.6	－	－
		送信成功応答のタイムアウトモニタリング		4.3.1.6	－	－
		Inter-ECU (COM)		4.3.1.6	－	●
		Intra-ECU (RTE)		4.3.1.6	－	－
		送信ステータスの取得		4.3.1.6	－	●
		Explicit status read access (Last-is-best semantics)	Rte_Send	4.3.1.6	－	●
		Explicit status read access (queue semantics)	Rte_Write	4.3.1.6	－	●
		Implicit status read access (Last-is-best semantics)	Rte_IStatus	4.3.1.6	－	●
SR8	Communication Time-out	データ受信のタイムアウト監視		4.3.1.7	－	●
		タイムアウト時間の指定		4.3.1.7	●	●
		受信タイムアウトモニタリング（Inter-ECU）		4.3.1.7	●	●
		受信タイムアウトのイベント通知		4.3.1.7	－	●
		ステータスの取得（Implicit 連携）	Rte_IStatus	4.3.1.7	－	●
SR9	Data Element Invalidation	データの無効化		4.3.1.8	●	●
		直接アクセスによるデータ無効化 （即時にデータ無効化）	Rte_Invalidate	4.3.1.8	●	●
		間接アクセスによる無効化 （runnable 処理後にデータ無効化）	Rte_IInvalidate Rte_IStatus	4.3.1.8	－	●
		データ無効化オプション（Sender）		4.3.1.8	●	●
		無効化の有無指定 （can Invalidate）	Rte_Invalidate Rte_IInvalidate	4.3.1.8	●	●

			Rte_IStatus			
		データ無効化オプション（Receiver）		4.3.1.8	●	●
		無効として扱う（keep）		4.3.1.8	●	●
		初期値に変換（replace）		4.3.1.8	●	●
SR10	Filters	データ受信フィルタの指定		4.3.1.9	●	●
SR11	Buffering	Last-is-Best-Semantics for ‘data’ Reception (バッファなし)	Rte_Read	4.3.1.10	●	●
		Queueing for ‘event’ Reception （バッファあり）	Rte_Receive	4.3.1.10	●	●
SR12	Data types	Primitive Data Types		4.3.1.11.1.1		●
		Complex Data Types		4.3.1.11.1.2	–	●
	Fan-out	Fan-out (Inter-ECU : “PDU fan-out”)		4.3.1.11.3	●	●
		Fan-out (Inter-ECU : “Signal fan-out”)		4.3.1.11.3	–	●
		Fan-out (Intra-ECU)		4.3.1.11.3	●	●
	Fan-in	Fan-in (Inter-ECU : “PDU fan-in”)		4.3.1.11.4	●	●
		Fan-in (Inter-ECU : “Signal fan-in”)		4.3.1.11.4	–	●
		Fan-in (Intra-ECU)		4.3.1.11.4	●	●

6.1.5. Client-Server 連携

ID	分類	機能	RTE API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	ﾌﾟﾛﾌｧｲﾙ S1
Client-Server						
CS1	Introduction	同期 Client-Server 連携		4.3.2.1	●	●
		ECU 内連携（Direct Call）	Rte_Call	4.3.2.1	●	●
		ECU 内連携	Rte_Call	4.3.2.1	–	–
		ECU 間連携	Rte_Call	4.3.2.1	–	–
CS2	Introduction	非同期 Client-Server 連携		4.3.2.1	–	●
		ECU 内連携	Rte_Call Rte_Result	4.3.2.1	–	●
		ECU 間連携	Rte_Call Rte_Result	4.3.2.1	–	●
CS3	Multiplicity	n : 1 の Client-Server 連携サポート		4.3.2.2.1	●	●
		n オペレーション/ポートのサポート		4.3.2.2.2	●	●
		Server リクエストのキューイング		4.3.2.2.4	–	●
CS4	Communication Time-out	同期サーバ呼び出しのタイムアウトモニタリング		4.3.2.3	–	–
		ECU 内連携		4.3.2.3	–	–
		ECU 間連携		4.3.2.3	–	–

		非同期サーバ呼び出しタイムアウトモニタリング		4.3.2.3	–	–
		ECU 内連携		4.3.2.3	–	–
		ECU 間連携		4.3.2.3	–	●
CS5	Port-Defined argument values	ポート定義引数のサポート (Port API Option#PORT-ARG-VALUES)		4.3.2.4	●	●
CS6	Buffering	同期 Client-Server のリクエスト		4.3.2.5	–	–
		非同期 Client-Server のリクエスト／レスポンス		4.3.2.5	–	●
CS7	Data Mapping	Inter-ECU Mapping		4.3.2.7.1	–	●
		Intra-ECU Mapping		4.3.2.7.1		●
CS8	Fault detection and reporting	Inter-ECU Mapping		4.3.2.7.1	–	●
		Intra-ECU Mapping		4.3.2.7.1	●	●

6.1.6. コンポーネント内部連携

ID	分類	機能	RTE API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	ﾌﾟﾛﾌｧｲﾙ S1
SWC Internal Communication						
IR1	InterRunnableVariables	Runnable 間の共通変数 (直接アクセス)	Rte_IrvRead Rte_IrvWrite	4.3.3.1	●	●
		Runnable 間の共通変数 (間接アクセス)	Rte_IrvIRead Rte_IrvIWrite	4.3.3.1	●	●
IR2	Initial Values	InterRunnableVariable の初期値定義		4.3.3.1	●	●

6.1.7. モード

ID	分類	機能	RTE API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	ﾌﾟﾛﾌｧｲﾙ S1
Mode						
MD1	Mode User	現在の Mode 取得	Rte_ModeType Rte_Mode	4.4.1	-	-
MD2	Mode User	Mode 変更イベントによる runnable 起動		4.4.1	-	-
		Mode 変更による runnable の起動		4.4.1	-	-
		依存関係に応じた runnable の起動		4.4.1	-	-
MD3	Mode Manager	Mode の変更		4.4.2	-	-
		Mode 変更(non-blocking)	Rte_Switch	4.4.2	-	-
		Mode 変更(blocking)	Rte_Switch (Rte_Feedback)	4.4.2	-	-

MD4	Order of actions taken by the RTE upon interception of a mode switch notification	ModeDeclaratioinGroup で定義された モード状態遷移管理		4.4.4	-	-
		モードに応じた runnable の実行制御		4.4.4	-	-

6.1.8. 初期化と終了処理

ID	分類	機能	RTE API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Initialization and Finalization						
IF1	Initialization and Finalization of the RTE	RTE の初期化／終了	Rte_Start Rte_Stop	4.5.1	●	●
IF2	Initialization and Finalization of AUTOSAR Software-Components	ソフトウェアコンポーネントの初期化／終了		4.5.2	-	-

6.2. RTE API リファレンス

本ツールで生成可能な RTE モジュールでサポートするデータ型、および API について記載する。

【表記方法】

表記シンボル	シンボルの意味
●	AUTOSAR RTE 仕様に準拠する。
▲	部分制約。AUTOSAR RTE 仕様に対して制約あり
×	未サポート
ー	未サポート（JASPAR プロファイル機能に含まれない）

6.2.1. RTE API Data Types

○Data Type

分類	API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	S1
Primitive AUTOSAR Data Types	CHAR-TYPE (uint8)	5.3.4.2	●	●
	CHAR-TYPE (uint16)	5.3.4.2	–	–
	STRING-TYPE (uint8[n])	5.3.4.2	–	●
	STRING-TYPE (uint8 *)	5.3.4.2	–	–
	INTEGER-TYPE (sint8)	5.3.4.2	●	●
	INTEGER-TYPE (sint16)	5.3.4.2	●	●
	INTEGER-TYPE (sint32)	5.3.4.2	●	●
	INTEGER-TYPE (uint8)	5.3.4.2	●	●
	INTEGER-TYPE (uint16)	5.3.4.2	●	●
	INTEGER-TYPE (uint32)	5.3.4.2	●	●
	OPAQUE-TYPE (uint8)	5.3.4.2	●	●
	OPAQUE-TYPE (uint16)	5.3.4.2	●	●
	OPAQUE-TYPE (uint32)	5.3.4.2	●	●
	REAL-TYPE (float32)	5.3.4.2	–	●
	REAL-TYPE (float64)	5.3.4.2	–	●
	BOOLEAN-TYPE (boolean)	5.3.4.2	●	●
Complex AUTOSAR Data Types	ARRAY-TYPE (Primitive Data Type elements)	5.3.4.3	–	●
	ARRAY-TYPE (Complex Data Types elements)	5.3.4.3	–	–
	ARRAY-TYPE (Primitive and Complex Type elements)	5.3.4.3	–	–
	RECORD-TYPE(Primitive Data Type member)	5.3.4.3	–	●
	RECORD-TYPE(Complex Data Type member)	5.3.4.3	–	–
	RECORD-TYPE(Primitive and Complex Type member)	5.3.4.3	–	–

○Standard Return Type

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Std ReturnType	Infrastructure Errors	5.5.1.1	●	●
	Immediate Infrastructure Errors	5.5.1.1	●	●
	Overlayed Errors	5.5.1.1	●	●
	Application Errors	5.5.1.2	●	●
	Immediate Infrastructure Errors	5.5.1.2	●	●
	Overlayed Errors	5.5.1.2	●	●
	Predefined Error Codes	5.5.1.3	●	●

○Rte Instance Handle

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Rte Instance	Rte_Instance	5.5.2	–	–

○RTE Macros

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
RTE Modes	Rte_ModeType_<ModeDeclarationGroup>	5.5.3	–	–
	RTE_TRANSITION_<ModeDeclarationGroup>	5.5.3	–	–
	RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>	5.5.3	–	–

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Enumeration Data Types	<EnumLiteral>	5.5.4	●	●

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Range Data Types	<DataType>_LowerLimit <DataType>_UpperLimit	5.5.5	●	●

6.2.2. RTE APIs (Communication)

■Sender-Receiver Communication

分類	API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	S1
Sender-Receiver communication APIs (Last-is-best semantics / explicit communication APIs)	Rte_Read_<p>_<o>	5.6.8	●	●
	Rte_Read_<p>_<o> [Blocking]	5.6.8	–	–
	Rte_Write_<p>_<o>	5.6.4	●	●
	Rte_Invalidate_<p>_<o>	5.6.6	●	●
	Rte_Feedback_<p>_<o>	5.6.7	–	●
	Rte_Feedback_<p>_<o> [Blocking]	5.6.7	–	–
Sender-Receiver communication APIs (Last-is-best semantics / implicit communication APIs)	Rte_IRead_<re>_<p>_<d>	5.6.15	–	●
	Rte_IWrite_<re>_<p>_<d>	5.6.16	–	●
	Rte_IWriteRef_<re>_<p>_<d>	5.6.17	–	●
	Rte_IInvalidate_<re>_<p>_<d>	5.6.18	–	●
	Rte_IStatus_<re>_<p>_<d>	5.6.19	–	●
Sender-Receiver communication APIs (queue semantics / explicit communication APIs)	Rte_Receive_<p>_<o>	5.6.9	●	●
	Rte_Receive_<p>_<o> [Blocking]	5.6.9	–	–
	Rte_Send_<p>_<o>	5.6.4	●	●
	Rte_Feedback_<p>_<o>	5.6.7	–	●
	Rte_Feedback_<p>_<o> [Blocking]	5.6.7	–	–

■Client-Server Communication

分類	API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	S1
Client-Server communication APIs(synchronous call)	Rte_Call_<p>_<o>	5.6.10	●	●
Client-Server communication APIs (asynchronous call)	Rte_Call_<p>_<o>	5.6.10	–	●
	Rte_Result_<p>_<o>	5.6.11	–	●
	Rte_Result_<p>_<o> [Blocking]	5.6.11	–	–

■SWC Internal Communication

分類	API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	S1
InterRunnableVariable APIs (explicit communication APIs)	Rte_IrvRead_<re>_<name>	5.6.22	●	●
	Rte_IrvWrite_<re>_<name>	5.6.23	●	●
InterRunnableVariable APIs (implicit communication APIs)	Rte_IrvIRead_<re>_<name>	5.6.20	●	●
	Rte_IrvIWrite_<re>_<name>	5.6.21	●	●

■Mode

Mode 機能をサポートしないため、Mode API は提供しない

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Mode APIs	Rte_Switch_<p>_<o>	5.6.5	–	–
	Rte_Feedback_<p>_<o>	5.6.7	–	–
	Rte_Feedback_<p>_<o> [Blocking]	5.6.7	–	–
	Rte_Mode_<p>_<o>	5.6.26	–	–

6.2.3. RTE APIs (Behavior)

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Exclusive Area APIs	Rte_Enter_<name>	5.6.24	●	●
	Rte_Exit_<name>	5.6.25	●	●

S0／S1：

JASPAR 拡張 API として、Runnable 実行 API をサポートする。

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Runnable Invoke API [Jaspar Extension]	Rte_Invoke_<c>_<re>	–	●	●

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Per-Instance Memory APIs	Rte_Pim_<name>	5.6.12	–	–

6.2.4. RTE APIs (Measurement & Calibration)

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Measurement & Calibration APIs	Rte_CData_<name>	5.6.13	–	–
	Rte_Calprm_<p>_<name>	5.6.14	–	–

6.2.5. RTE APIs (Indirect)

分類	API	Sec.# of RTE SWS	プロファイル S0	プロファイル S1
Indirect API	Rte_Ports_<i>_<R>	5.6.1	–	–
	Rte_NPorts_<i>_<R/P>	5.6.2	–	–
	Rte_Port_<p>	5.6.3	–	–

6.2.6. RTE Events

イベント種類	イベント動作	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ	
			S0	S1
TimingEvent	Activate runnable entities	5.7.5.1	●	●
	Wake up runnable entities	5.7.5.1	–	●
	Activation concurrently	5.7.5.1	–	●
ModeSwitchEvent	Activate runnable entities	5.7.5.1	–	–
	Wake up runnable entities	5.7.5.1	–	–
	Activation concurrently	5.7.5.1	–	–
AsynchronousServerCallReturnsEvent	Activate runnable entities	5.7.5.1	–	●
	Wake up runnable entities	5.7.5.1	–	●
	Activation concurrently	5.7.5.1	–	●
DataReceiveErrorEvent	Activate runnable entities	5.7.5.1	–	●
	Wake up runnable entities	5.7.5.1	–	●
	Activation concurrently	5.7.5.1	–	●
OperationInvokedEvent	Activate runnable entities	5.7.5.1	●	●
	Wake up runnable entities	5.7.5.1	–	●
	Activation concurrently	5.7.5.1	●	●
DataReceivedEvent	Activate runnable entities	5.7.5.1	–	●
	Wake up runnable entities	5.7.5.1	–	●
	Activation concurrently	5.7.5.1	–	●
DataSendCompletedEvent	Activate runnable entities	5.7.5.1	–	●
	Wake up runnable entities	5.7.5.1	–	●
	Activation concurrently	5.7.5.1	–	●

6.2.7. RTE Lifecycle APIs

分類	API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ	
			S0	S1
RTE Lifecycle APIs	Rte_Start	5.9.3	●	●
	Rte_Stop	5.9.3	●	●

6.2.8. RTE Call-backs

分類	API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ	
			S0	S1
Communication Service Call-backs	Rte_COMCbk_<sn>	5.9.3	●	●

	Rte_COMCbkTAck_<sn>	5.9.3	–	●
	Rte_COMCbkTErr_<sn>	5.9.3	–	●
	Rte_COMCbkInv_<sn>	5.9.3	●	●
	Rte_COMCbkTOut_<sn>	5.9.3	●	●
	Rte_COMCbk_<sg>	5.9.3	●	●
	Rte_COMCbkTAck_<sg>	5.9.3	–	●
	Rte_COMCbkTErr_<sg>	5.9.3	–	●
	Rte_COMCbkInv_<sg>	5.9.3	–	●
	Rte_COMCbkTOut_<sg>	5.9.3	–	●

6.2.9. VFB Tracing APIs

分類	API	Sec.# of RTE SWS	ﾌﾟﾛﾌｧｲﾙ S0	S1
RTE API Trace Events	Rte_<api>Hook_<c>_<ap>_Start	5.10.2.1.1	●	●
	Rte_<api>Hook_<c>_<ap>_Return	5.10.2.1.2	●	●
COM Trace Events	Rte_ComHook_<signalName>_SigTx	5.10.2.2.1	●	●
	Rte_ComHook_<signalName>_SigRx	5.10.2.2.2	●	●
	Rte_ComHook_<signalName>_SigIv	5.10.2.2.3	●	●
	Rte_ComHook<Event>_<signalName>	5.10.2.2.4	●	●
OS Trace Events	Rte_Task_Activate	5.10.2.3.1	●	●
	Rte_Task_Dispatch	5.10.2.3.2	●	●
	Rte_Task_SetEvent	5.10.2.3.3	●	●
	Rte_Task_WaitEvent	5.10.2.3.4	●	●
	Rte_Task_WaitEventRet	5.10.2.3.5	●	●
Runnable Entity Trace Events	Rte_Runnable_<c>_<reName>_Start	5.10.2.4.1	●	●
	Rte_Runnable_<c>_<reName>_Return	5.10.2.4.2	●	●

6.3. RTE Generator ツール

以下では、本ツールで提供する RTE 生成機能について説明する。

【表記方法】

表記シンボル	シンボルの意味
●	AUTOSAR RTE 仕様に準拠する。
▲	部分制約。AUTOSAR RTE 仕様に対して制約あり
×	未サポート
ー	未サポート（JASPAR プロファイル機能に含まれない）

6.3.1. RTE 生成プロセス

RTE Generation Process に関する提供機能

機能	フェイズ	プロファイル	
		S0	S1
Component API 生成	Contract phase	●	●
RTE 生成	Generation phase	●	●
Os コンフィグレーション生成 (OsNeeds 生成)	Generation phase	—	—

RTE 構成モジュールの生成機能

生成対象ファイル	C 言語ファイル名	プロファイル	
		S0	S1
RTE Header File	Rte.h	●	●
Life Cycle Header File	Rte_Main.h	●	●
Application Header File	<App>.h	●	●
AUTOSAR type Header File	Rte_Type.h	●	●
VFB Tracing Header File	Rte_Hook.h	●	●
RTE Configuration Header File	Rte_Cfg.h	●	●
RTE Scheduler File （※）	Rte_Scheduler.c, Rte_Scheduler.h ほか	●	—

※RTE Scheduler File
S0 プロファイルで、コマンドラインから -sch が指定された場合のみ生成される。

6.3.2. RTE コンフィグレーション

以下では､本リリースツールで対応する RTE コンフィグレーション項目について説明する。

ID	分類	設定項目	設定内容	Sec.# of RTE SWS	プロファイル	
					S0	S1
RTE Generation Parameters						
	RteGeneration	RteCalibrationSupport	データ較正指定	6.1	–	–
		RteGenerationMode	RTE 生成モード	6.1	●	●
			互換モード	6.1	●	●
			ベンダモード	6.1	●	●
		RteMeasurementSupport	データ測定指定	6.1	–	–
		RteOptimizationMode	RTE 最適化モード	6.1	×	×
			MEMORY	6.1	×	×
			RUNTIME	6.1	×	×
		RteVfbTrace	VFBTrace 指定	6.1	●	●
		RteVfbTraceFunction	VFBTrace 指定	6.1	●	●
		JasparProfile （※1）	JASPAR プロファイル		●	●
Handling of Software Component instances						

	SwComponentInstance	[ImplementationRef]	実装モデル定義	6.2	●	●
		[ServiceComponentPrototypeRef]	ｺﾝﾎﾟｰﾈﾝﾄ指定	6.2	–	–
		[SoftwareComponentInstanceRef]	ｺﾝﾎﾟｰﾈﾝﾄ指定	6.2	●	●
	ExclusiveAreaImplementation	ExclusiveAreaImplMechanism	実装ﾒｶﾆｽﾞﾑ指定	6.2	–	●
		[ExclusiveAreaRef]	排他エリア指定	6.2	–	●
	NVRamAllocation	RamBlockLocationSymbol0..1		6.2	–	–
		RomBlockLocationSymbol0..1		6.2	–	–
		[SwNvRamMappingReference]		6.2	–	–
		[NvmBlockRef]		6.2	–	–
	RunnableEntityMapping	ActivationOffset	実行オフセット	6.2	●	●
		TimingEvent		6.2	●	●
		TimingEvent 以外		6.2	–	–
		PositionInTask	タスク内実行順	6.2	●	●
		同種イベント／タスク内の実行順番定義		6.2	●	●
		別種イベント／タスク内の実行順番定義		6.2	–	–
		[RTEEventRef]	動作イベント	6.2	●	●
		[MappedToTaskRef]	動作コンテキスト指定	6.2	●	●
		Runnable の動作タスク定義		6.2	●	●
		Runnable の動作割り込み（ISR）定義（※2）		–	●	●
		呼出し元タスクでの動作定義（同期 Client-Server 連携の Server Runnable）	（タスク指定なし）	6.2	●	●
		呼出し元タスクでの動作定義（Sender-Receiver 連携の Runnable）	（タスク指定なし）	6.2	–	–
		[UsedOsEventRef]	制御イベント	6.2	–	●
Component Type Calibration						
	ComponentTypeCalibration	CalibrationSupportEnabled	較正指定	6.3	–	–
		[ComponentTypeRef]	較正ｺﾝﾎﾟｰﾈﾝﾄ	6.3	–	–

※1:JasparProfile は、JASPAR 仕様で定義されたコンフィグレーション項目である。
適用する JASPAR プロファイル（FULL,S1,S0,S0OS）を指定する。

※2:MappedToTaskRef 項目による割り込み（ISR）動作定義は、AUTOSAR 標準 RTE には含まれない JASPAR 拡張仕様である。

7. 変更履歴詳細

7.1. Version1.1.0 (2012.12.07)

VFB トレーシング、および IRV 対応版リリース。

A) RTE モジュール差分

■機能追加

１）SW-C 間の内部変数機能対応
InterRunnableVariable 機能をサポート対象機能に追加した。

２）VFB トレーシング対応
VFB トレーシング機能をサポート対象機能に追加した。

■不具合対応

１）Rte_Feedback API の API 名誤りの不具合修正
（対応前）Rte_FeedBack_<p>_<d>
（対応後）Rte_Feedback_<p>_<d>

B) RTE Generator ツールの差分

■機能追加

１）RTE 生成オプションの追加
ツールの起動パラメータを追加し、生成した RTE の内部データを保護するために利用する OS API の選択肢を追加した。

（対応前）：以下の２つのどちらかを選択可能
・SuspendAllInterrupts/ ResumeAllInterrupts
・EnableAllInterrupts /DisableAllInterrupts

（対応後）：以下の３つのうちいずれかを選択可能
・SuspendAllInterrupts/ ResumeAllInterrupts
・EnableAllInterrupts /DisableAllInterrupts
・SuspendOSInterrupts/ ResumeOSInterrupts

２）利用可能な AUTOSAR XML 仕様の拡大
AUTOSAR 3.2 仕様の XML データを読み込んで RTE モジュールを生成可能に対応。
（ただし、RTE モジュールがサポートする機能範囲は、AUTOSAR R3.0 範囲となる）

7.2. Version1.0.5 (2009.12.10)

S0 プロファイル対応 JASPAR OS の対応版リリース。

A) RTE モジュール差分
なし

B) RTE Generator ツールの差分
■機能変更
１）ECU 設計工程で、ＲＴＥプロファイルとして S0 プロファイル（OS なし）が指定されていた場合に、RTE Scheduler モジュールが生成されないように修正した。

補足：
S0 プロファイル（OS なし）の場合は、JASPAR OS0 を適用するため、
RTE Scheduler は不要となる。

２）RTE 生成オプションの追加
ツールの起動パラメータを追加し、S0 プロファイル用の汎用スケジューラモジュール「RTE Scheduler」の生成の有無を指定可能なようにした。
（追加）：-sch　　S0 プロファイルの場合のみ有効。-sch が指定されると RTE Scheduler を生成する。

補足：
本オプションは、旧互換性を保持するために残している。
通常は、JASPAR OS0 と組み合わせるため RTE Scheduler は不要となる。

■不具合対応
１）Opaque 型の定数リテラルの不具合対応
Opaque 型データ値の初期値、、無効値として 1 バイトデータのリテラルを指定すると 10 進数で認識され、16 進数で指定できない不具合の対応。
１バイトのデータを扱う場合、リテラルの最後に":"を追加すること 16 進数値で指定とできるように修正した。

参考：リテラル設定例：
- １バイトデータ(10 進数) 10 を設定する場合のリテラル
  10
- １バイトデータ(16 進数) 0x10 を設定する場合のリテラル（今回より指定可能）
  10:
- ２バイトデータ(16 進数) 0xFF, 0x11 を設定する場合のリテラル
  FF:11

7.3. Version1.0.4 (2009.8.28)

JPTC (JASPAR Tool Chain) 環境対応版リリース

A) RTE モジュール差分

■不具合修正

1）Rte_IRead API の仕様誤り修正
レコード型のデータ取得 API の戻り値誤りを修正
（修正前） RecordType の変数を返す
（修正後） RecordType の変数のポインタを返す。

例：レコード型（MyRecord 型）のデータ値の取得 API の場合
（修正前）データ値の実体を返す。
MyRecord Rte_IRead_Runnable_port_value();

（修正後）データ値へのポインタを返す。
MyRecord* Rte_IRead_Runnable_port_value();

■性能改善

1）消費 RAM 量の低減
JASPAR プロファイルへのモジュール構造最適化を進め、消費 ROM/RAM 量を低減。

B) RTE Generator ツールの差分

■機能追加

1）JPTC（JASPAR Tool Chain - Eclipse 環境）対応
JPTC の画面上からジェネレータを起動する機能に対応。

2）RTE 生成オプションの追加
ツールの起動パラメータを追加し、生成した RTE の内部データを保護するために
利用する OS API を選択可能なようにした。

（対応前）：SuspendOSInterrupts/ ResumeOSInterrupts 固定

（対応後）：以下のどちらかを選択可能
・SuspendOSInterrupts/ ResumeOSInterrupts
・EnableAllInterrupts /DisableAllInterrupts

■性能改善

1）ツール処理時間の改善
ツールの内部処理を見直し、コード生成に必要な時間を約半分に短縮。

7.4. Version1.0.3 (2009.5.29)

JASPAR プロファイルツールチェーン対応版リリース

A) RTE モジュール差分
■性能改善
１）消費 RAM 量の低減
JASPAR プロファイルへのモジュール構造最適化を進め、消費 RAM 量を低減。

２）処理時間の改善
初期化処理に必要となる処理時間を短縮。

B) RTE Generator ツールの差分
■機能追加

１）JASPAR プロファイルのツールチェーン対応
RTE コンフィグレータで指定した JASPAR プロファイルに応じた
RTE モジュールを生成するように対応。

２）RTE モジュールの生成オプション対応
RTE コンフィグレータで指定した RTE モジュール生成オプション
（COMPATIBILITY_MODE ／ VENDOR_MODE）
に対応した RTE モジュールを生成するように対応。

・COMPATIBILITY_MODE（R1.0.2J までのサポートモード）
AUTOSAR 互換モードでモジュールを生成する。
AUTOSAR 仕様に対して、ソースコード互換／オブジェクトコード互換の
RTE モジュールを生成する。

・VENDOR_MODE
ベンダモードでモジュールを生成する。
AUTOSAR 仕様に対して、ソースコード互換の RTE モジュールを生成する。
バイナリ互換用の仕様を未サポートとしてモジュール構造を最適化しており、
COMPATIBILITY_MODE で生成するモジュールと比較して、
RAM 消費量が低減される。

7.5. Version1.0.2 (2009.4.9)

JASPAR RTE 1.0 版 性能改善版ツールリリース

A) RTE モジュール差分
■問題・不具合の対応
１）[性能改善]

S1 プロファイルにおいて、プロファイルへの最適化を進め消費 RAM 量を改善。

２）[性能改善]
S0、S1 プロファイルにおいて、RTE モジュールの内部構造のインライン化を行い処理速度を高速化し、かつ消費スタック量を改善。

B) RTE Generator ツールの差分
■問題・不具合の対応
１）ECU コンフィグレーションにて、インプリメンテーションに利用するコンポーネントの内部動作を指定しても正しく反映されない不具合を修正。

7.6. Version1.0.1 (2009.2.13)

JASPAR RTE 1.0 版ツールリリース

7.7. Version1.0.0 (2008.12.29)

JASPAR RTE β版ツールリリース

A) RTE モジュール差分
■問題・不具合の対応
１）Sender-Receiver 連携で、データ送信タイムアウトの COM 通知ハンドリング方法を COM 仕様に合わせて変更
（変更前）送信エラー／送信タイムアウトは、Rte_COMCbkTErr_<sn/sg> コールバックを利用して RTE 側で検出。
（変更後）送信エラーは、Rte_COMCbkTErr_<sn/sg> コールバックを利用して RTE 側で検出。
送信タイムアウトエラーは、Rte_COMCbkTOut_<sn/sg> コールバックを利用して RTE 側で検出。

B) RTE Generator ツール差分
■問題・不具合の対応
１）Sender-Receiver 連携データに対してマッピング定義が複数あった場合に COM コンフィグレーション内容が正しく反映されない不具合を修正。

２）レコード型／配列型データの初期値／無効値のリテラルが定義されていた場合、定数リテラルのシンボル名称が一致せず、コンパイルエラーになる不具合を修正。