NCES

次世代車載システム向け RTE
外部仕様書

Ver.1.4.0

2016/12/27

名古屋大学大学院情報科学研究科附属 組込みシステム研究センター

＜目次＞

1. 概要

1.1　本文書の目的

本文書は RTE および SCHM の機能仕様を規定するものである．

本仕様は，AUTOSAR Specification of RTE で規定される仕様(以降，AUTOSAR 仕様と略す)をベースに，必要な拡張と修正を行ったものである．

本文書は，RTE および SCHM に関する一般的な知識を持ったソフトウェア技術者が読むことを想定して記述している．AUTOSAR 仕様に関する知識があることが望ましいが，それを前提とせず記述している．

1.2　関連文書

1.2.1　ベースとした文書

以下の表は，本文書のベースとする文書であり，その内容は本文書内に包含されている．

文書名	バージョン
AUTOSAR Specification of RTE	V3.2.0 (R4.0 Rev 3)

1.2.2　参考文書

以下の表は，本文書から参照している文書，または本文書を理解するために必要な文書である．内容は本文書に包含されていない．

文書名	バージョン
ATK2 外部仕様書	1.2.1
次世代車載システム向け COM 外部仕様書	1.1.0
Software Component Template	V4.2.0 (R4.0 Rev 3)
BSW Module Description Template	V2.2.0 (R4.0 Rev 3)
System Template	V4.2.0 (R4.0 Rev 3)
Specification of ECU Configuration	V3.2.0 (R4.0 Rev 3)
Specification of Memory Mapping	V1.4.0 (R4.0 Rev 3)
Specification of Compiler Abstraction	V3.2.0 (R4.0 Rev 3)
OSEK/VDX Communication	V3.0.2 (R4.0 Rev 3)

1.3　凡例

1.3.1　仕様番号

本文書では，AUTOSAR 仕様と，名古屋大学大学院情報科学研究科附属組込みシステム研究センター(NCES)を中心とする次世代車載システム向け RTOS の仕様検討及び開発に関するコンソーシアム型共同研究(ATK2/AP コンソーシアム)で新規に規定した仕様が混在しているため，以下に示す仕様番号を用いてこれらの仕様を区別して管理を行う．仕様番号は，要求事項にのみ付与することを基本とする．ただし，AUTOSAR 仕様において，概念の説明や補足事項についても仕様番号が付与されているものに関しては，そのまま付与する．また，本文書から ATK2/AP コンソーシアムで開発した A-RTE および A-SCHM の実装に依存する仕様を，参考情報として本文書に記載するため，仕様番号を付与する．

仕様番号	内容
【rte_sws_xxxx】	関連文書 AUTOSAR Specification of RTE で規定された仕様．AUTOSAR 仕様で記述されている RTE 仕様番号を用いる．本仕様に採用しなかった AUTOSAR 仕様についても，【rte_sws_xxxx】で記載している．
【rte_sws_ext_xxxx】	関連文書 AUTOSAR Specification of RTE で規定された外部モジュール要求仕様．RTE/SCHM を使用するその他のモジュール(SW-C, BSWM, ECU インテグレーションコード等)への要求事項を規定する．AUTOSAR 仕様で記述されている RTE 仕様番号を用いる．本仕様番号に反した使い方を使われた場合，別途規定がない限りは，本 RTE/SCHM は，動作を保証しない．
【rte_sws_a_xxxx】	関連文書 AUTOSAR Specification of RTE に記述されているが，仕様番号表記がないもの．AUTOSAR 仕様の RTE/SCHM への要求事項であるため，本仕様では，仕様番号を付与する．
【rte_sws_ext_a_xxxx】	関連文書 AUTOSAR Specification of RTE に記述されているが，仕様番号表記がないもの．AUTOSAR 仕様の外部モジュールへの要求事項であるため，本仕様では，仕様番号を付与する．
【nrte_sws_xxxx】	ATK2/AP コンソーシアムで新規に規定した RTE/SCHM 仕様．
【nrte_sws_ext_xxxx】	ATK2/AP コンソーシアムで新規に規定した外部モジュール要求仕様．
【irte_sws_xxxx】	ATK2 の実装において規定した仕様．【rte_sws_xxxx】【nrte_sws_xxxx】において，実装定義と規定されている仕様や，その他の仕様だけでは実装が不明確である場合に，参考情報として本文書に記載する．
【rte_sws_xxxx_Conf】 【nrte_sws_xxxx_Conf】	AUTOSAR 仕様でコンフィギュレーション情報に関して記述された仕様．また，ATK2/AP コンソーシアムで新規に規定したコンフィギュレーション情報に関して記述された仕様に【nrte_sws_xxxx_Conf】を付与する．
〔rte_sws_xxxx〕 〔nrte_sws_xxxx〕	本文中で上記の仕様番号を参照する際に使用する． 仕様定義である【rte_sws_xxxx】【nrte_sws_xxxx】と区別して表記する．

1.3.2　コンテナ，およびパラメータ

本文書では，RTE/SCHM のコンフィギュレーション方法を示すため，各コンフィギュレーションで指定するコンテナ，およびパラメータを以下のように記載する．

表記	内容
コンテナ／パラメータ和名 **(コンテナ／パラメータ名)**	RTE/SCHM の ECU コンフィギュレーション情報のコンテナ，もしくはパラメータを表す．詳細については，3.14 節を参照．
コンテナ／パラメータ和名 *(コンテナ／パラメータ名)*	RTE/SCHM の ECU コンフィギュレーション情報以外のコンテナ，もしくはパラメータを表す．詳細については，以下の文書を参照． ・　関連文書「次世代車載システム向け RTOS 外部仕様書」 ・　関連文書「次世代車載システム向け COM 外部仕様書」 ・　関連文書「Software Component Template」 ・　関連文書「BSW Module Description Template」 ・　関連文書「System Template」 ・　関連文書「Specification of ECU Configuration」

*ショートネーム(shortName)*は本仕様書上で多用されるため，例外として，カッコ内を省略して「ショートネーム」と記載する．

1.3.3　注記

AUTOSAR 仕様との違い

削除や改変を行った AUTOSAR 仕様に対して，どのような差分があるかを説明する．

使用上の注意

本仕様に準拠した RTE/SCHM の開発者ではなく，本 RTE/SCHM を使用してアプリケーション開発を行うユーザに対する，注意事項もしくは推奨事項を説明する．

<特定要素>の設定

本 RTE/SCHM の特定要素に対して ECU インテグレーションを行うユーザに対する，RTE/SCHM のコンフィギュレーション方法を説明する．

サポート範囲の制限

開発プロジェクトの計画等の理由により，AUTOSAR 仕様のサポート範囲を制限する事項を説明する．

2. 概念

2.1　機能一覧

2.1.1　RTE 機能一覧

本 RTE が提供する機能の概略を以下に示す.

ランナブル動作管理

- ランナブルのライフサイクル(起動，開始，および終了)の管理

ランナブル周期起動

- ランナブルの一定周期，およびオフセットでの起動

ランナブルバックグラウンド起動

- バックグラウンド動作のためのランナブルの循環起動

S/R 連携(Sender/Receiver 連携)

- データの送信，および無効化
- 送信 ACK
- データの受信
- 受信データのタイムアウト監視
- 受信データのフィルタリング

C/S 連携(Client/Server 連携)

- サービスの呼出し

ランナブル間連携

- データの送信
- データの受信

排他エリア

- SW-C の排他制御

RTE ライフサイクル管理

- RTE の開始，および終了
- パーティションの停止，および再起動

2.1.2　SCHM 機能一覧

本 SCHM が提供する機能の概略を以下に示す.

BSW スケジューラブル動作管理

- BSW スケジューラブルのライフサイクル(起動，開始，および終了)の管理

BSW スケジューラブル周期起動

- BSW スケジューラブルの一定周期，およびオフセットでの起動

BSW スケジューラブルバックグラウンド起動

- バックグラウンド動作のための BSW スケジューラブルの循環起動

モード連携

- SCHM のモード連携

排他エリア

- BSWM の排他制御

SCHM ライフサイクル管理

- SCHM の開始，および終了

2.2　主要概念

2.2.1　SW-C(ソフトウェアコンポーネント)

車載システムの機能の一部を提供するソフトウェア部品であり，車載システムの機能は，複数の SW-C の機能を組み合わせることで実現される．

SW-C 間の連携は RTE を経由して行われる．

2.2.2　BSWM(ベーシックソフトウェアモジュール)

ECU の基盤機能を提供するソフトウェア部品である．本 RTE は，以下の BSWM を使用する．

- OS
- COM

BSWM の提供する機能は，RTE を経由して SW-C に提供される．

SCHM は，以下の BSWM を使用する【rte_sws_7519】.

- OS

2.2.3　RTE ジェネレータ

RTE/SCHM は，車載システム内の ECU 毎に必要な機能を実現するため，ツールによってソースコードが生成される．このツールを RTE ジェネレータ(RTEGEN)と呼ぶ.

RTEGEN は，以下の 2 つの処理から RTE/SCHM のソースコードを生成する．以下のフェーズに関する詳細は，3.1.5 節を参照.

- コントラクトフェーズ
- ジェネレーションフェーズ

ECU コンフィギュレーション情報

車載システム内の ECU 毎に，RTE，および OS や COM といった BSWM をコンフィギュレーションするための設計情報．RTEGEN の入力となる.

本書では，1.3.2 の表記に基づき，ECU コンフィギュレーション情報の設計内容と RTE の機能仕様との関連を示す.

ECU 抽出システムコンフィギュレーション情報

車載システムを構成する SW-C の内，1 つの ECU に関する情報のみ抽出した設計情報．RTEGEN の入力となる.

本書では，1.3.2 の表記に基づき，ECU 抽出システムコンフィギュレーション情報の設計内容と RTE の機能仕様との関連を示す.

BSWM ディスクリプション情報

車載システム内の ECU 毎に，BSWM の排他エリア等をコンフィギュレーションするための設計情

報．RTEGEN の入力となる．

本書では，1.3.2 の表記に基づき，BSWM ディスクリプション情報の設計内容と SCHM の機能仕様との関連を示す．

RTE コード

RTEGEN によって生成される RTE のソースコード．RTEGEN の出力となる．

SCHM コード

RTEGEN によって生成される SCHM のソースコード．RTEGEN の出力となる．

2.2.4　ECU インテグレーションコード

AUTOSAR で標準化されない ECU 固有の機能を提供するソフトウェアである．ECU インテグレーションにおいて実装される．

ECU インテグレーションコードが提供する主な機能として以下のものがある．

- ECU 固有の起動／終了処理．
- ECU 固有のエラー処理(OS 保護違反の処理等)．

2.2.5　ランナブル

ランナブルは，SW-C の機能を実現するための処理であり，ランナブルはそれぞれ 1 つの対応する C 言語の関数(以下，エントリポイント関数)を持つ．

ランナブルは，何らかの RTE イベントを起動契機として起動される．

2.2.6　BSW スケジューラブル

BSW スケジューラブルは，BSWM の機能を実現するための処理である．BSW スケジューラブルは，それぞれ対応する 1 つのエントリポイント関数を持つ．

BSW スケジューラブルは，何らかの BSW イベントを起動契機として起動される．

2.2.7　RTE イベント

RTE イベントはランナブルの起動契機であり，RTE イベントの条件を満たす際に，対応するランナブルを起動する．

あるランナブルに対し RTE イベントを複数割り当てることで，複数の起動契機からランナブルを起動することができる．

2.2.8　BSW イベント

BSW イベントは BSW スケジューラブルの起動契機であり，BSW イベントの条件を満たす際に，対応する BSW スケジューラブルを起動する．

ある BSW スケジューラブルに対し BSW イベントを複数割り当てることで，複数の起動契機から

BSW スケジューラブルを起動することができる．

2.2.9　BSWM エンティティ

BSWM エンティティは，BSWM の以下の処理のスーパークラスである．

- BSW スケジューラブル(BswSchedulableEntity)
- BSW 直接起動処理(BswCalledEntity)
- BSW 割込み処理(BswInterruptEntity)

BSW 直接起動処理

BSW 直接起動処理は，他 BSW から直接起動される処理である．

BSW 割込み処理

BSW 割込み処理は，割込みを契機として起動される処理である．

2.2.10 エクスキュータブル

エクスキュータブルは，RTE/SCHM 上で動作する以下の処理のスーパークラスである．

- ランナブル
- BSWM エンティティ

エクスキュータブル実行インスタンス

エクスキュータブルの実行毎のインスタンスを表す．複数のエクスキュータブルが並行実行される際は，処理の呼出し毎にインスタンスとして区別する．

2.2.11 コア

本 RTE/SCHM では，シングルコアシステム，およびマルチコアシステムをサポートする．
RTE/SCHM は，以下の 2 つの種類のプロセッサコア上での動作をサポートする．プロセッサコアの定義については「次世代車載システム向け RTOS 外部仕様書」を参照．

コア種別	説明
マスタコア	システム起動時に唯一起動するコアである．
スレーブコア	他のコアから起動されるコアである．

BSWM の動作するコア

本 RTE/SCHM は，OS，および COM が，以下のコア上で動作することを期待する．

BSWM	動作コア	説明
OS	マスタコア，および全てのスレーブコア	OS 機能がいずれのコアからも使用可能であることを期待する【nrte_sws_ext_0018】.
COM	マスタコア	マルチコアに対応していない COM を使用する場合，本 RTE は，以下を期待する【nrte_sws_ext_0015】. ・ COM がマスタコア上で動作し，マスタコア上から使用可能であること. ・ COM 機能がマスタコア以外のコアから使用されないこと.
	マスタコア，および全てのスレーブコア	マルチコアに対応している COM を使用する場合，本 RTE は，以下を期待する【nrte_sws_ext_0033】 ・ COM がマスタコア，および全てのスレーブコア上で動作し，全てのコア上で使用可能であること.

<u>コアの設定</u>

本 RTE/SCHM は，*OsOS(OsOS)*の*コア数(OsNumberOfCores)*が定義されている場合，システムがマルチコアシステムであるものとして扱い，*コア数(OsNumberOfCores)*が定義されていない場合，システムがシングルコアシステムであるものとして扱う【nrte_sws_0159】.

マスタコアは，*OsOS(OsOS)*の*マスタコア ID(OsMasterCoreId)*により指定する【nrte_sws_0160】.

2.2.12 パーティション

パーティションは ECU を機能単位に分割する論理的な保護境界である．パーティションには，SW-C と BSWM の両方を所属させることができる．パーティションの導入により，異なるパーティションに所属するソフトウェアを互いに保護することができる.

パーティションを使用する ECU をパーティション構成の ECU，パーティションを使用しない ECU を非パーティション構成の ECU と呼ぶ.

マルチコアシステムにおいて，パーティションはいずれか 1 つのコアにマッピングされる.

<u>パーティションの権限</u>

パーティションは権限の違いから以下の 2 つに分類される.

種別	説明
信頼パーティション	全パーティションの OS オブジェクト，およびメモリオブジェクトにアクセス可能である.
非信頼パーティション	自分自身のパーティション内の OS オブジェクト，およびメモリオブジェクトにのみアクセス可能である.

BSWM 配置パーティション

あるコアにマッピングされる全 BSWM は，そのコア内の 1 つの信頼パーティションにマッピングしなければならない【nrte_sws_ext_0001】．このパーティションを BSWM 配置パーティションと呼ぶ.

信頼コンテキスト

信頼 OSAP に所属する OS オブジェクトのコンテキスト(OS タスク，もしくは OS ISR).

パーティションの実現方式

RTE/SCHM は，パーティションを OSAP の使用によって実現する．信頼パーティションは信頼 OSAP，非信頼パーティションは非信頼 OSAP を使用して実現する【rte_sws_a_0001】.

パーティション構成の設定

本 RTEGEN は，*パーティションコレクション(EcucPartitionCollection)*が定義されている場合，ECU がパーティション構成であるものとして扱い，*パーティションコレクション(EcucPartitionCollection)*が定義されていない場合，ECU が非パーティション構成であるものとして扱う【nrte_sws_0161】.

パーティションの設定

パーティションは，*パーティション(EcucPartition)*により指定する【rte_sws_a_0002】.

パーティションの権限は，*パーティション(EcucPartition)*を参照する *OSAP(OsApplication)*の *OS 権限(OsTrusted)*により指定する【nrte_sws_0002】.

パーティションがいずれのコアにマッピングされるかは，*パーティション(EcucPartition)*を参照する *OSAP(OsApplication)*の *OS コア割当て(OsCoreAssignment)*により指定する【nrte_sws_0182】.

BSWM 配置パーティションの設定

RTEGEN は，*パーティション(EcucPartition)*の *BSWM 実行パーティション(EcucPartitionBswModuleExecution)*が true に設定されたパーティションを，BSWM 配置パーティションとして扱う【rte_sws_a_0003】.

2.3　BSWM から SW-C への割込み通知

発生した割込みに応じて SW-C の動作を行う必要がある場合，SW-C への割込み通知に変換する必要がある．SW-C は割込みを直接扱うことができないため，この変換は BSWM において行われる．

SW-C への割込み通知とは，ランナブルが割込みにより起動され(ランナブルの起動元の OS タスクが起動される，もしくは OS イベントが設定される)，それぞれの割込みにおける固有のデータを受け取ることを意味する．

2.3.1　割込み通知の種別

SW-C への割込み通知の実現手段として，以下の２つの方法があり，どちらの方法が提供されるかは，BSWM の種類に依存する．

- 標準インタフェース
- AUTOSAR インタフェース

2.3.1.1　標準インタフェースによる割込み通知

この通知は OS，および COM により提供され，RTE により間接的に SW-C への割込み通知が行われる．

OS, および COM が提供するコールバックにより RTE への通知が行われる．RTE は，この通知をランナブルの RTE イベントに変換し，SW-C への割込み通知を実現する．

RTE は，COM コールバックによるランナブルの起動をサポートする【rte_sws_3531】．

2.3.1.2　AUTOSAR インタフェースによる割込み通知

AUTOSAR サービスを提供する BSWM，もしくは CDD-C が割込みを処理し，AUTOSAR インタフェースの呼出しに変換することで，SW-C への割込み通知を実現する．

<u>SW-C(CDD-C)による割込み処理</u>

割込みのコンテキストで SW-C を実行してはならない【rte_sws_ext_a_0001】．ただし，CDD-C に限り，ISR の実現と，割込みへの直接的な関与を許可する【rte_sws_ext_a_0002】．

2.3.2　割込みカテゴリによる RTE 機能の呼出し制限

C1ISR からは，RTE にアクセスしてはならない【rte_sws_ext_7816】．

2.4　SW-C 間の連携

2.4.1　連携の構成

SW-C 間の連携は以下の要素から構成される．

ポートインタフェース

SW-C 間の連携内容を規定するインタフェース仕様．

インタフェース要素

SW-C 間の連携単位．ポートインタフェース内で複数定義される．SW-C 間のデータの送受信，およびサービスの呼出し等の連携はインタフェース要素毎に独立して行われる．

提供側ポート

連携を提供する SW-C のポート．

要求側ポート

連携の開始を要求する SW-C のポート．

提供側 SW-C

提供側ポートを保持し，ポートインタフェースを介した連携を提供する SW-C．

要求側 SW-C

要求側ポートを保持し，ポートインタフェースを介した連携の開始を要求する SW-C．

2.4.2　連携の種別

以下の 2 種類の連携をサポートする．

種別	説明
S/R 連携	SW-C 間でデータの送受信を行う．本連携の詳細は 2.8 節を参照．
C/S 連携	SW-C 間でサービスの呼出しを行う．本連携の詳細は 2.9 節を参照．

2.4.2.1　連携のパターン

RTE は，異なるパーティション間の通信や，異なる ECU 間の通信を想定する．RTE により，ECU コンフィグ情報に従った連携が実現され【rte_sws_2200】，AUTOSAR SW-C ディスクリプションにより与えらえた連携の属性のセマンティックが実現される【rte_sws_2201】．また，センサやアクチュエータ SW-C と ECU 抽象のポート間の連携は，AUTOSAR SW-C のポート間の連携と全く同じ作法で生成される【rte_sws_2050】．S/R 連携においてサポートされる連携のパターンは，2.8 節に詳細を記載する．C/S 連携においてサポートされる連携のパターンは，2.9 節に詳細を記載する．

以下に，SW-C における連携のパターンを示す．

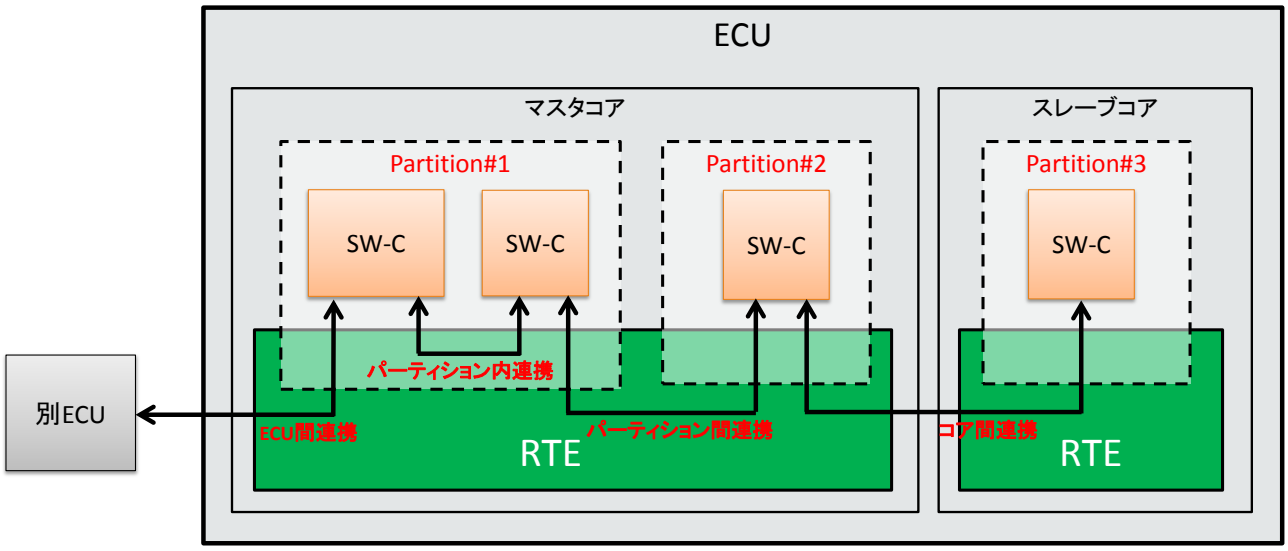

図 2-1 連携のパターン

パーティション内連携

1 つの ECU，1 つのパーティション内における連携をパーティション内連携と呼ぶ．

パーティション間連携

1 つの ECU 内における異なるパーティション間の連携をパーティション間連携と呼ぶ．パーティションは，異なるコアに配置される場合や，ECU 内の異なるメモリ領域に配置される場合もある．

コア間連携

パーティション間連携のうち，マルチコアシステムにおける異なるコア間の連携となるものをコア間連携と呼ぶ．

ECU 間連携

異なる ECU 間の連携を ECU 間連携と呼ぶ．一般に，ECU 間連携は COM により実現される．

ECU 内連携

1 つの ECU 内での連携を総称して ECU 内連携と呼ぶ．これは，パーティション内連携と，パーティションが ECU 内の異なるメモリ領域に配置された場合のパーティション間連携が該当する．

2.4.2.2　連携の多重度

SW-C 間の連携では，以下で説明する提供側ポートと要求側ポートの多重度が想定される．S/R 連携においてサポートされる多重度は，2.8 節に詳細を記載する．C/S 連携においてサポートされる多重度

は，2.9 節に詳細を記載する．

1:0 連携，あるいは 0:1 連携

RTE は，要求側ポートと提供側ポートが接続されていない場合でも動作可能とする【rte_sws_1329】．このような連携を 1:0 連携，あるいは 0:1 連携と呼ぶ．

1:1 連携

単一の提供側ポートと単一の要求側ポート間におけるデータの送受信，あるいはサービスの呼出しを行うことを 1:1 連携と呼ぶ．以下に，S/R 連携，および C/S 連携の 1:1 連携のイメージを示す．

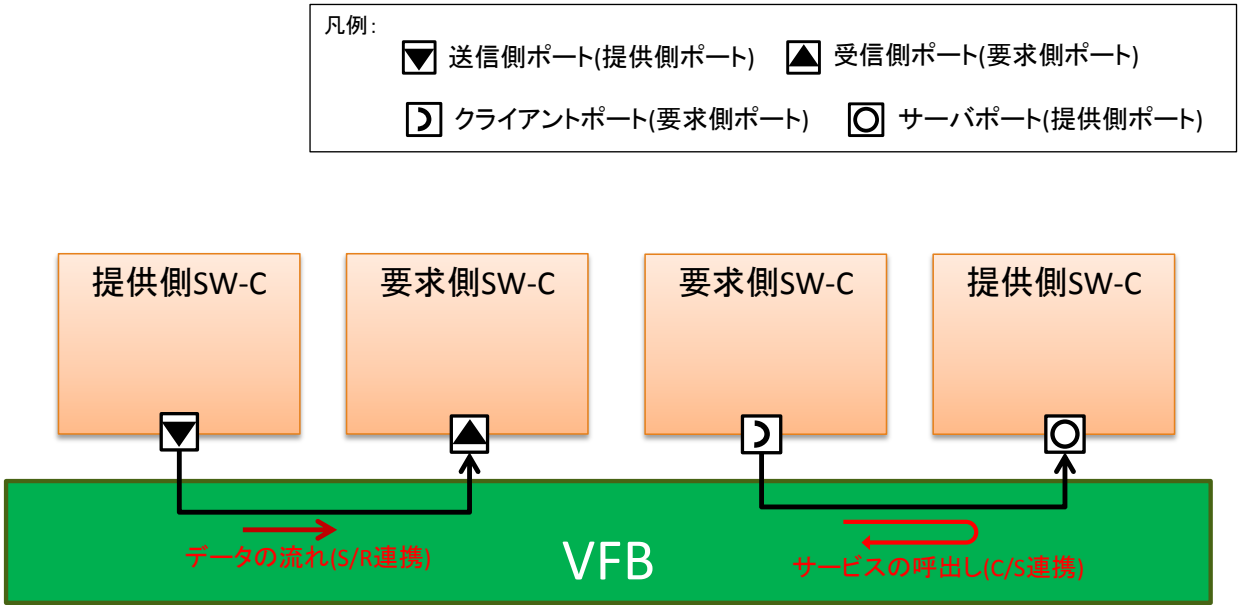

図 2-2 S/R 連携，および C/S 連携の 1:1 連携

1:N 連携

S/R 連携の場合，単一の提供側ポートから複数の要求側ポートに対してデータの送受信を行うことを 1:N 連携と呼ぶ．以下に，S/R 連携における 1:N 連携のイメージを示す．

C/S 連携の場合，単一の要求側ポートから複数の提供側ポートに対してサービスの呼出しを行うことに相当するが，C/S 連携の 1:N 連携を RTE はサポートしない．

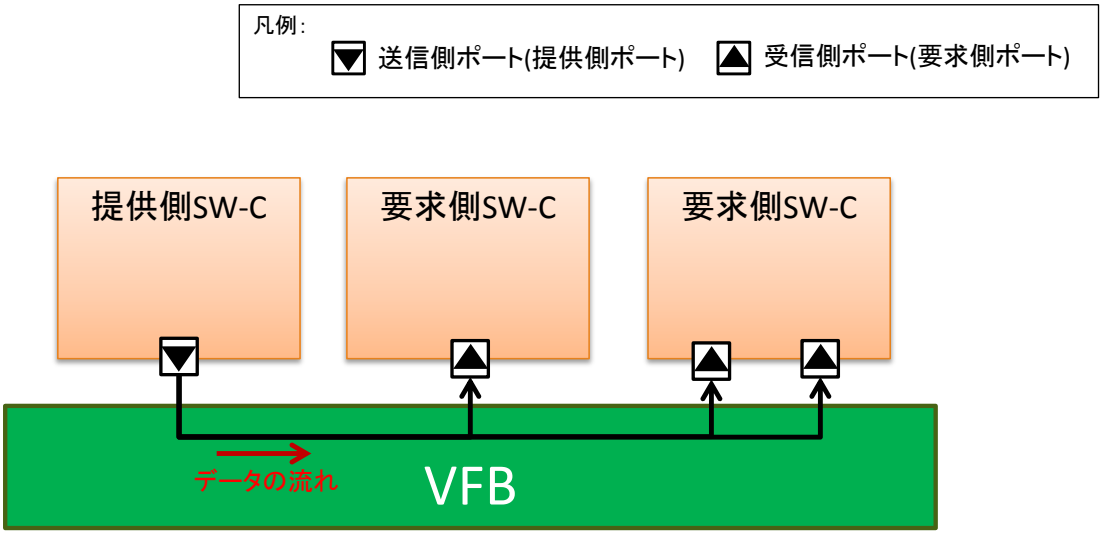

図 2-3 S/R 連携における 1:N 連携

N:1 連携

S/R 連携の場合，複数の提供側ポートから単一の要求側ポートに対してデータの送受信を行うことに相当し，S/R 連携の N:1 連携も RTE はサポートする．

C/S 連携の場合，複数の要求側ポートから単一の提供側ポートに対してサービスの呼出しを行うことを N:1 連携と呼ぶ．以下に，C/S 連携における N:1 連携のイメージを示す．

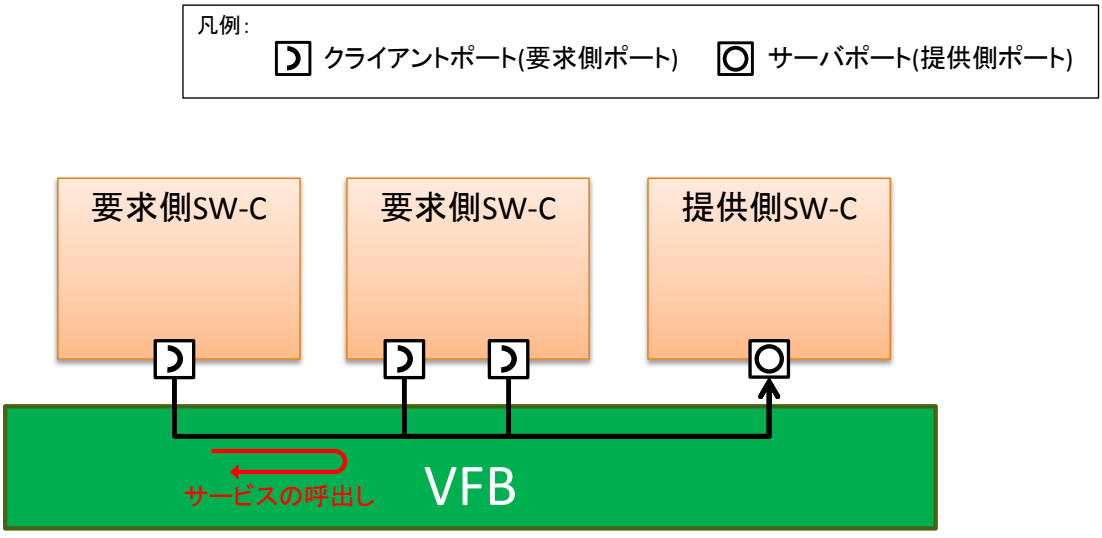

図 2-4 C/S 連携における N:1 連携

N:M 連携

S/R 連携の場合，複数の提供側ポートと複数の要求側ポートの間でデータの送受信を行うことを N:M 連携と呼ぶ．以下に，S/R 連携における N:M 連携のイメージを示す．

C/S 連携の N:M 連携を RTE はサポートしない．

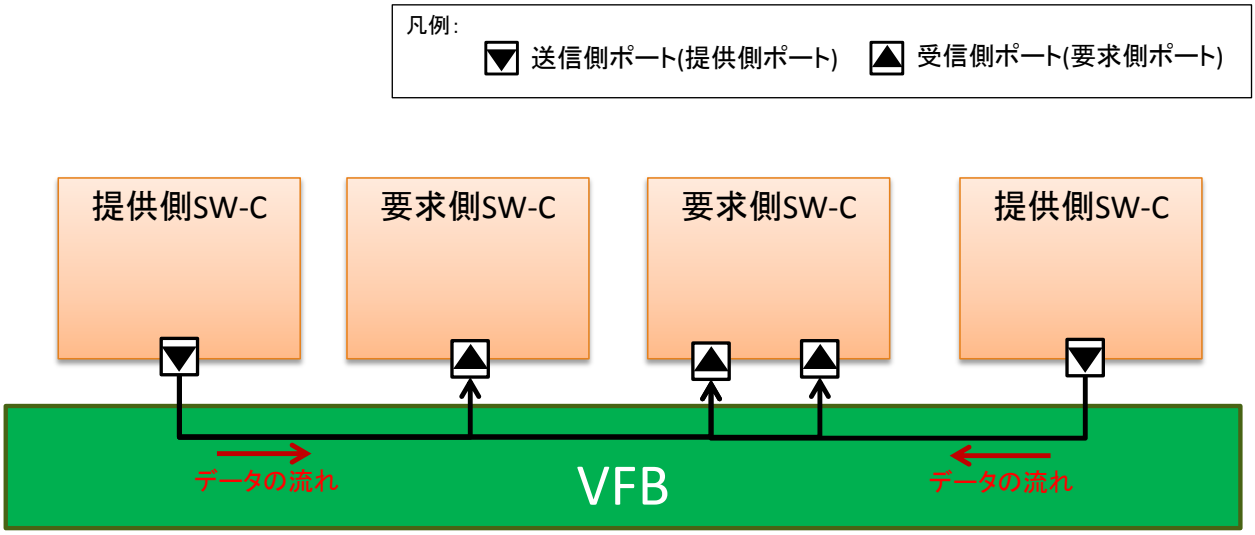

図 2-5 S/R 連携における N:M 連携

2.4.3　連携のデータ一貫性

RTE は，AUTOSAR インタフェースを通じた連携について，データ一貫性を保証する【rte_sws_3514】．

2.4.4　連携の実現方式

2.4.4.1　パーティション内連携

パーティション内の連携については，RTEGEN は，RTE モジュール内部で連携を実装するか，COM を使用して連携を実装するかを選択することができる．

ECU 内連携でありながら，インタフェース要素に COM シグナル/COM シグナルグループがマッピングされている場合，RTEGEN は送信に COM シグナル/COM シグナルグループを使用することができる．また，これを無視して RTE 内で直接データをやりとりしてもよい【rte_sws_a_0004】．本 RTEGEN では，パーティション内連携は COM を使用せず，RTE 内部で実現する【nrte_sws_0003】.

2.4.4.2　パーティション間連携

RTEGEN は，マイクロコントローラに依存しない RTE コードの生成機能をサポートしなければならない【rte_sws_2734】．そのため，RTE は OS から提供される機能を使用してパーティション間連携を実現する．

本 RTE は，パーティション間のデータの送受信の実現のために，IOC または OS 信頼関数を使用する【nrte_sws_0004】．IOC と OS 信頼関数のどちらの方式を採用するかは実装定義とする【nrte_sws_0253】.

IOC を使用する場合の制約および期待する仕様

IOC は 1:1 連携，および N:1 連携のみをサポートするため，本 RTE では，1:N 連携は，RTE が受信側毎に対応する IOC 用システムサービスを呼び出すことで実現する【nrte_sws_0005】．

本 RTE は，IOC 用システムサービスにおいて，並行アクセスを防止することを期待する【nrte_sws_ext_0006】.

IOC のコールバックは割込みのコンテキストで動作するため，RTE はエクスキュータブルを IOC コールバックのコンテキストで実行してはならない【rte_sws_2736】．

OS 信頼関数を使用する場合の制約および期待する仕様

OS 信頼関数内で直接呼び出すエクスキュータブルは，信頼パーティションに所属していなければならない【rte_sws_7606】.

信頼パーティションは停止または再起動されないことを期待する【rte_sws_ext_a_0009】.

OS 信頼関数に渡される全ての情報は，信頼関数内の先頭において，呼び出し元パーティションに所属することをチェックしなければならない【nrte_sws_0378】.

RTE API の参照渡しされる引数のチェック

非信頼パーティションの SW-C が OUT 引数または IN 引数(複合データ型の場合)を受け取る際，RTE は，参照渡しされたポインタの全メモリ領域が呼び出し元のパーティションに所属することが確認できた場合のみ，信頼パーティションの SW-C へそのポインタを渡す【rte_sws_2752】．

非信頼パーティションの SW-C が IN 引数(ポインタ実装データ型の場合)を受け取る際，RTE は，値渡しされたポインタの先頭アドレスが呼び出し元のパーティションに所属することが確認できた場合のみ，信頼パーティションの SW-C へそのポインタを渡す【rte_sws_2753】．

以上の要件を満たすため，RTE は，OS の CheckTaskMemoryAccess を使用する【rte_sws_a_0005】．

AUTOSAR 仕様との違い

AUTOSAR 仕様では，IOC 用システムサービスはリエントラントではないため，RTE は，異なるエクスキュータブル実行インスタンスからの同一の IOC 用システムサービスへの並行アクセスを防止しなければならないと規定されている【rte_sws_2737】．しかし，OS で提供されるシステムサービスは通常リエントラントで実行できるように実装されていることが想定される．したがって，並行アクセス防止に要するオーバヘッドを最小限にするため，IOC 用システムサービスにおいて並行アクセスを防止することを前提とし〔nrte_sws_ext_0006〕，本 RTE では，RTE による IOC 用システムサービスへの並行アクセスの防止は実施しない【nrte_sws_0166】．

2.4.4.3　ECU 間連携

RTE は，ECU 間連携を実現するために，COM を使用する【rte_sws_a_0006】.

COM を使用する必要がある場合，COM との連携を行うのは RTE の責任である【rte_sws_8306】. ECU 間連携を実施するコアの種別，およびパーティションの権限により，COM の機能呼出しのために必要な手順が異なる.

マスタコアからの ECU 間連携

マスタコア，かつ信頼パーティションからの ECU 間連携について，RTE は，COM API を直接呼び出す【nrte_sws_0006】.

マスタコア，かつ非信頼パーティションからの ECU 間連携について，RTE は，信頼パーティションに配置される COM API を OS 信頼関数経由で呼び出す【rte_sws_a_0007】.

スレーブコアからの ECU 間連携

マルチコアに対応している COM を使用する場合，スレーブコア，かつ信頼パーティションからの ECU 間連携について，RTE は，COM API を直接呼び出す【nrte_sws_0334】. A-RTE は，データ受信の場合のみ COM API を使用する【irte_sws_0023】. データ送信の場合は，マルチコアに対応していない COM を使用する場合の RTE の実現方式を採用する【irte_sws_0024】.

マルチコアに対応していない COM を使用する場合，COM はマスタコアにマッピングされるため，スレーブコア上から COM API を直接呼び出すことは許されない.

スレーブコアからの ECU 間のデータ送信は，スレーブコアからのデータ送信要求を，マスタコア上の COM API 呼出しに変換することで実現する【nrte_sws_0007】. スレーブコアからマスタコアへのデータ送信要求は，IOC 用システムサービスにより実現する【nrte_sws_0008】.

スレーブコアへの ECU 間のデータ受信は，マスタコア上の COM コールバックの通知をスレーブコアへのデータ受信通知に変換することで実現する【nrte_sws_0009】. マスタコアからスレーブコアへのデータ受信通知は，RTE の内部バッファを使用することで実現する【nrte_sws_0321】.

スレーブコアからの ECU 間のデータ送信を実現するため，RTEGEN は，以下の 2 つの BSW スケジューラブルを生成する【rte_sws_8307】. これらの BSW スケジューラブルの処理内容は実装定義とする【nrte_sws_0011】.

エンティティ名	説明
Rte_ComSendSignalProxyPeriodic	周期的に送信する COM シグナル/COM シグナルグループの送信処理を行う.
Rte_ComSendSignalProxyImmediate	即座に送信する COM シグナル/COM シグナルグループの送信処理を行う.

A-RTEGEN は，Rte_ComSendSignalProxyPeriodic を生成する場合，本 BSW スケジューラブルを

起動するタスクは生成しない【irte_sws_0012】．Rte_ComSendSignalProxyPeriodic は，SCHM のコンフィギュレーションにより，周期実行設定する必要がある【nrte_sws_ext_0029】.

A-RTEGEN は，Rte_ComSendSignalProxyImmediate を生成する場合，本 BSW スケジューラブルを起動する専用タスク(Rte_ComSendSignalProxyImmediateTask)を合わせて生成する【irte_sws_0013】．Rte_ComSendSignalProxyImmediateTask は，OS のイベント待ちシステムサービスによって，実行要求を待つ方法で実現するため，ECU インテグレーションコードは，Rte_ComSendSignalProxyImmediateTask を最初に一度起動する必要がある【nrte_sws_ext_0030】.

使用上の注意

スレーブコアからの ECU 間通信の場合，別の OS タスクを経由した通信となり遅延が大きいため，使用を推奨しない.

2.4.4.4　連携の返り値

RTE API における返り値の伝搬は，IOC 用システムサービスによる実現か，COM API 呼出しによる実現かをコンフィギュレーションにより選択可能としなければならない【rte_sws_8308】.

AUTOSAR 仕様との違い

RTE API における返り値の伝搬を COM API 呼出しにより実現すると遅延が大きいため，本 RTEGEN では，本コンフィギュレーションをサポートせず，常に IOC 用システムサービスにより実現する【nrte_sws_0057】.

2.5　エクスキュータブル動作管理

RTE は，ランナブル動作のライフサイクル管理を行う．SCHM は，BSW スケジューラブル動作のライフサイクル管理を行う．ランナブルと BSW スケジューラブルは共に同じスーパークラスであるエクスキュータブル（2.2.10 節を参照）に属しているため，ランナブル動作と BSW スケジューラブル動作のライフサイクル管理は共通点が多い．よって，本節では，共通する仕様はエクスキュータブルとして説明し，仕様差分がある場合のみ個別に説明する．

2.5.1　エクスキュータブル動作の構成

エクスキュータブル動作は以下の要素から構成される．

エクスキュータブル実行順番

エクスキュータブルのマッピング先の OS タスク内において，ランナブルが何番目に実行されるかを示す数値．

2.5.2　エクスキュータブル動作の種別

2.5.2.1　ランナブル動作の種別

ランナブルの起動契機は，以下のいずれかの RTE イベントである【rte_sws_2203】．

RTE イベント名	説明
周期イベント	設定された起動オフセット，および起動周期で，周期的にランナブルが起動する． 起動の行われるタイミングについては，2.6 節を参照．
バックグラウンドイベント	バックグラウンドでの起動を想定したランナブルを起動する． 起動の行われるタイミングについては，2.7 節を参照．
データ送信完了イベント	S/R 連携のデータの送信成功/失敗時にランナブルが起動する． 起動の行われるタイミングについては，2.8 節を参照．
データ受信イベント	S/R 連携のデータの受信時にランナブルが起動する． 起動の行われるタイミングについては，2.8 節を参照．
データ受信エラーイベント	S/R 連携の受信におけるエラー発生時にランナブルが起動する． 起動の行われるタイミングについては，2.8 節を参照．
オペレーション呼出しイベント	C/S 連携のサービスの呼出しに伴い，ランナブルが起動する． 起動の行われるタイミングについては，2.9 節を参照．

複数のランナブルのサポート

RTE は，複数の AUTOSAR SW-C に含まれる複数のランナブルをサポートする【rte_sws_2202】.

複数の RTE イベントによる起動

RTE は，異なる種類の，複数の RTE イベントによる，同じランナブルの起動をサポートする【rte_sws_3520】【rte_sws_3524】．また，これらの RTE イベントが，それぞれ異なる OS タスクにマッピングされることをサポートする〔nrte_sws_0016〕．ただし，後述する制約に注意すること．

複数の RTE イベントによる起動の制約

エントリポイント関数のシグネチャは，そのランナブルの起動契機となる RTE イベントにより決まる（3.6.5 節を参照）．RTE イベントが定めるシグネチャが異なる場合があるため，異なるシグネチャを持つ RTE イベントを 1 つのランナブルに割り当てると，シグネチャを定めることができない．したがって，以下の制約を定める．

異なるシグネチャを持つ RTE イベントを 1 つのランナブルに割り当てた場合，RTE は動作を保証しない【rte_sws_a_0008】．

ランナブル動作の可能なコンテキスト

ランナブルは，割込みからの OS タスク起動等，割込みの結果として実行されることは許されるが，割込みのコンテキストにおける実行は許されない〔rte_sws_ext_a_0001〕．ランナブルが OS タスクのコンテキスト以外から実行された場合，RTE は動作を保証しない【rte_sws_3600】．

ランナブルの多重起動

RTE は，ランナブルの並行起動によるエクスキュータブル実行インスタンスの多重起動をサポートする【rte_sws_3523】．

ランナブルの多重起動時のデータ一貫性保証

ランナブルが並行動作することによってデータ破壊される可能性を防ぐことは，SW-C 設計者の責任である【nrte_sws_ext_0003】．

2.5.2.2　BSW スケジューラブル動作の種別

BSW スケジューラブルの起動契機は，以下の BSW イベントである【rte_sws_7515】．

BSW イベント名	説明
周期イベント	設定された起動オフセット，および起動周期で，周期的に BSW スケジューラブルが起動する． 起動の行われるタイミングについては，2.6 節を参照．
バックグラウンドイベント	バックグラウンドでの起動を想定したランナブルを起動する． 起動の行われるタイミングについては，2.7 節を参照．
モード切替イベント	モード連携によるモードの状態遷移に伴い BSW スケジューラブルが起動する． 起動の行われるタイミングについては，2.11.4.1 節を参照．

複数の BSW イベントによる起動

SCHM は，異なる種類の，複数の BSW イベントによる，同じ BSW スケジューラブルの起動をサポートする【rte_sws_7526】【rte_sws_7527】．また，これらの BSW イベントが，それぞれ異なる OS タスクにマッピングされることをサポートする〔nrte_sws_0210〕．

BSW スケジューラブルの多重起動

SCHM は，BSW スケジューラブルの並行起動によるエクスキュータブル実行インスタンスの多重起動をサポートする【rte_sws_7525】．

BSW スケジューラブルの多重起動時のデータ一貫性保証

BSW スケジューラブルが並行動作することによってデータ破壊される可能性を防ぐことは，BSWM 設計者の責任である【nrte_sws_ext_0020】．

2.5.3　エクスキュータブル動作の状態

エクスキュータブル実行インスタンスは，以下に記述された状態遷移で振る舞う【rte_sws_2697】．

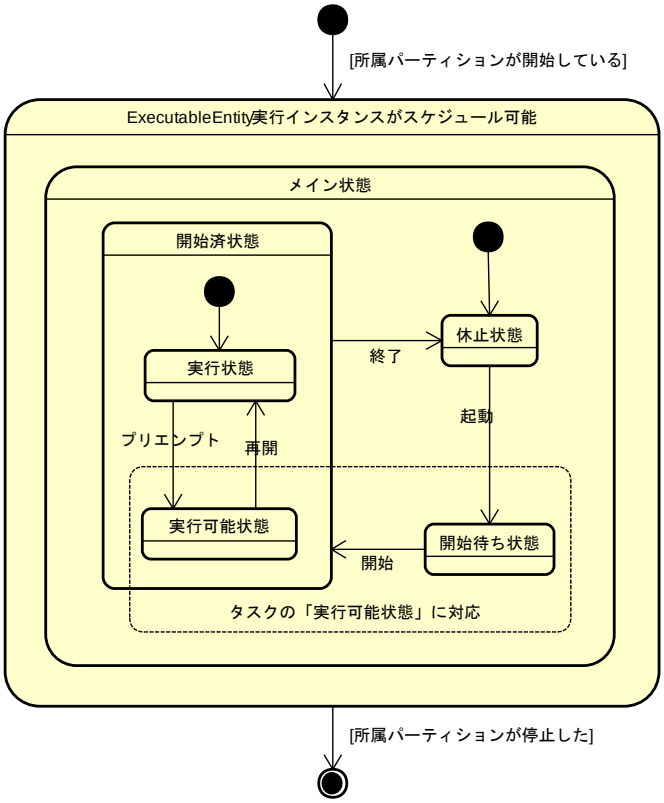

図 2-6 エクスキュータブル実行インスタンスの状態遷移

表 2-1 エクスキュータブル実行インスタンスの状態

エクスキュータブル 実行インスタンスの状態		説明
エクスキュータブル実行インスタンスがスケジュール可能		本状態はエクスキュータブル実行インスタンスのライフタイムを示している．エクスキュータブル実行インスタンスの状態遷移は，所属するパーティションが開始している場合のみ有効となる．
メイン状態	休止状態	エクスキュータブル 実行インスタンスは開始されておらず，エクスキュータブル 実行インスタンスへの開始要求もない．
	開始待ち状態	エクスキュータブル 実行インスタンスは起動されているが，動作を開始していない．
	実行状態	エクスキュータブルのエントリポイント関数が実行されている．
	実行可能状態	OS タスクが，エクスキュータブルのエントリポイント関数を実行中にプリエンプトされた．
	開始済状態	エクスキュータブルが開始され，終了されていない． 本状態は，「実行状態」，「実行可能状態」を含む親状態である．2.12 排他エリアにおけるエクスキュータブル実行時入退場は，本状態への入退場に相当する．

表 2-2 エクスキュータブル実行インスタンスの遷移

エクスキュータブル 実行インスタンスの遷移	説明（箇条書きはアクションの内容）
起動	エクスキュータブル実行インスタンスの開始要求があった．
開始	エクスキュータブル実行インスタンスのエントリポイント関数が呼び出された． ・ 2.12 節の排他エリア入退場方法で「エクスキュータブル実行時入退場」が有効な場合，排他エリアへの入場を行う．
プリエンプト	OS タスクの状態が実行状態から実行可能状態となった．
再開	OS タスクの状態が実行可能状態から実行状態となった．
終了	エクスキュータブル実行インスタンスの処理が完了した． ・ 2.12 節の排他エリア入退場方法で「エクスキュータブル実行時入退場」が有効な場合，排他エリアからの退場を行う．

表 2-3 エクスキュータブル実行インスタンスの状態遷移表

遷移 / エクスキュータブルの状態		起動	開始	プリエンプト	再開	終了
休止状態（初期状態）		開始待ち状態	－	－	－	－
開始待ち状態		－	実行状態	－	－	－
開始済状態	実行状態	－	－	実行可能状態	－	休止状態
	実行可能状態	－	実行状態	－	実行状態	休止状態

2.5.4　エクスキュータブル動作の実現方式

2.5.4.1　実現方式の構成要素

マッピング先 OS タスク

エクスキュータブルの動作に使用する OS タスク.

使用 OS イベント

エクスキュータブルの起動契機となる OS イベント.

タスクボディ

OS タスクが実行状態となった際に実行されるよう定義されたコード.

OS イベント待ち

OS のイベント待ちシステムサービスを要求し，指定された OS イベントが設定されるまで OS タスクが待ち状態となること.

2.5.4.2　ランナブル起動

ランナブル起動の実現方式は，以下のいずれかである．どの方式によりランナブルを起動するかは，コンフィギュレーションにより選択できる.

表 2-4 ランナブル起動の実現方式

ランナブル起動の実現方式	説明
OS タスク起動	OS タスクの起動によりランナブルを起動し，タスクボディからの関数呼出しによりランナブルを開始する．ランナブルはマッピング先 OS タスクのコンテキスト内で動作する． OS タスクの起動契機(OS アラームの満了等)で，OS タスク内にマッピングされる全てのランナブルの起動契機が実現できる場合に使用できる(図 2-7).
OS イベント設定	OS イベントの設定によりランナブルを起動し，タスクボディからの関数呼出しによりランナブルを開始する．ランナブルはマッピング先 OS タスクのコンテキスト内で動作する． OS イベントにより起動契機が識別できるため，OS タスクにマッピングされるランナブル間で起動契機が異なる場合にも使用できる(図 2-7).
直接関数起動	RTE API 内から直接関数呼出しにより，ランナブルを起動，および開始する．ランナブルは，RTE API の呼出し元ランナブルの，マッピング先 OS タスクのコンテキストで動作する．

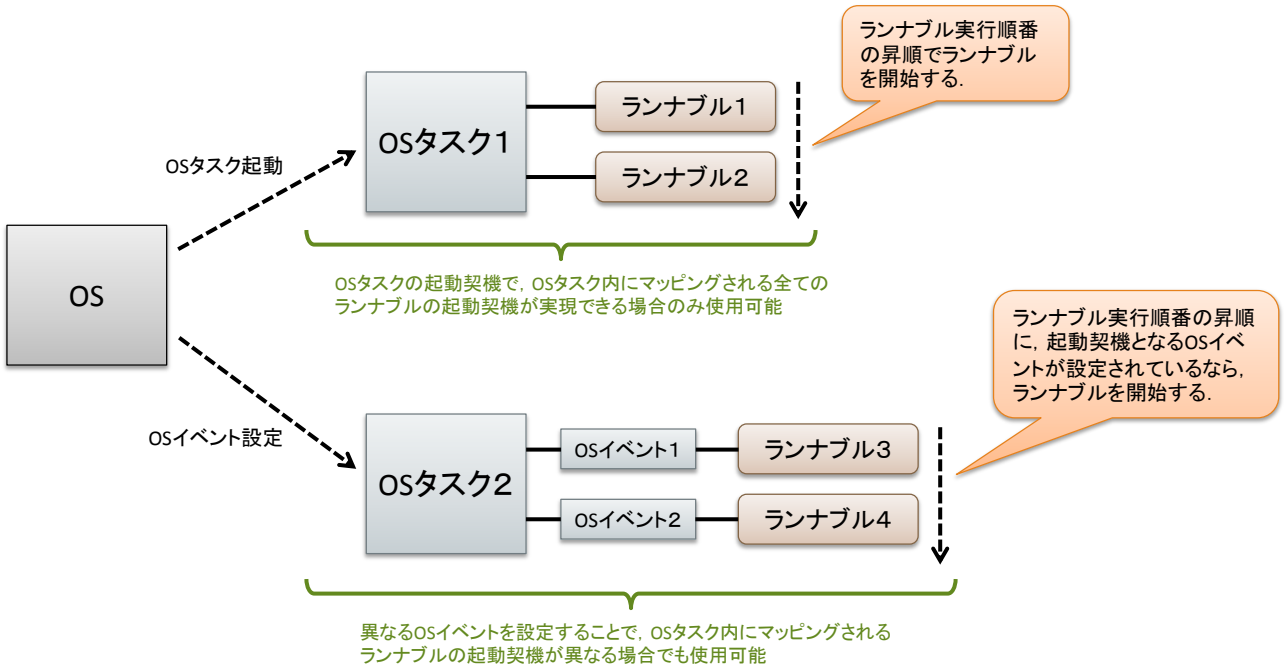

図 2-7 OS タスクにマッピングされたランナブルの起動

OS タスク起動の挙動

マッピング先 OS タスクが起動された場合，タスクボディから，OS タスクにマッピングされるランナブルを開始する(エントリポイント関数を呼び出す)【rte_sws_a_0009】．ある OS タスクから複数のランナブルが起動する場合，RTE は，エクスキュータブル実行順番の昇順で，ランナブルを開始する【rte_sws_a_0010】．RTEGEN は，設定された OS タスク内におけるランナブルの実行順番を尊重する【rte_sws_2207】.

OS イベント設定の挙動

マッピング先 OS タスクのタスクボディから，複数の使用 OS イベントに対する OS イベント待ちを要求する．いずれかの使用 OS イベントが設定された場合，使用 OS イベントが設定されているランナブルを開始する(エントリポイント関数を呼び出す)【rte_sws_a_0011】．ランナブルの実行が完了した後，OS タスクを終了させずに OS イベント待ちを再度要求する【rte_sws_a_0012】．

あるOSタスクから複数のランナブルが起動する場合，RTEは，エクスキュータブル実行順番の昇順で，ランナブルが開始するか(使用 OS イベントが設定されているか)の判定，および開始を順に行う【rte_sws_a_0013】．

直接関数呼出しの挙動

RTE API 内から，ランナブルを開始する(エントリポイント関数を呼び出す)【rte_sws_a_0014】．

ランナブルのライフサイクル管理

本 RTE の機能仕様範囲では，EcuStateManager および OS のライフサイクル管理により，RTE が行うべきランナブルのライフサイクル管理が実現可能である．このため，本 RTE では，ランナブルのライフサイクル管理を行わない【nrte_sws_0051】．

2.5.4.3　BSW スケジューラブル起動

BSW スケジューラブル起動の実現方式は，以下のいずれかである．どの方式により BSW スケジューラブルを起動するかは，コンフィギュレーションにより選択できる．

表 2-5 BSW スケジューラブル起動の実現方式

BSW スケジューラブル起動の実現方式	説明
OS タスク起動	OS タスクの起動により BSW スケジューラブルを起動し，タスクボディからの関数呼出しにより BSW スケジューラブルを開始する．BSW スケジューラブルはマッピング先 OS タスクのコンテキスト内で動作する． OS タスクの起動契機(OS アラームの満了等)で，OS タスク内にマッピングされる全ての BSW スケジューラブルの起動契機が実現できる場合に使用できる．
OS イベント設定	OS イベントの設定により BSW スケジューラブルを起動し，タスクボディからの関数呼出しにより BSW スケジューラブルを開始する．BSW スケジューラブルはマッピング先 OS タスクのコンテキスト内で動作する． OS イベントにより起動契機が識別できるため，OS タスクにマッピングされる BSW スケジューラブル間で起動契機が異なる場合にも使用できる．
直接関数起動	SCHM API 内から直接関数呼出しにより，BSW スケジューラブルを起動，および開始する．

OS タスク起動の挙動

マッピング先 OS タスクが起動された場合，タスクボディから，OS タスクにマッピングされる BSW スケジューラブルを開始する(エントリポイント関数を呼び出す)【rte_sws_a_0054】．ある OS タスクから複数の BSW スケジューラブルが起動する場合，SCHM は，エクスキュータブル実行順番の昇順で，BSW スケジューラブルを開始する【rte_sws_a_0055】．RTEGEN は，設定された OS タスク内における BSW スケジューラブルの実行順番を尊重する【rte_sws_7517】.

OS イベント設定の挙動

マッピング先 OS タスクのタスクボディから，複数の使用 OS イベントに対する OS イベント待ちを要求する．いずれかの使用 OS イベントが設定された場合，使用 OS イベントが設定されている BSW スケジューラブルを開始する(エントリポイント関数を呼び出す)【rte_sws_a_0056】．BSW スケジューラブルの実行が完了した後，OS タスクを終了させずに OS イベント待ちを再度要求する【rte_sws_a_0057】.

ある OS タスクから複数の BSW スケジューラブルが起動する場合，SCHM は，エクスキュータブル実行順番の昇順で，BSW スケジューラブルが開始するか(使用 OS イベントが設定されているか)の判定，および開始を順に行う【rte_sws_a_0058】.

直接関数呼出しの挙動

SCHM は，特定の条件を満たしている場合，SCHM API 内から，直接関数呼び出しにより BSW スケジューラブルを開始する(エントリポイント関数を呼び出す)．直接関数呼び出し条件の１つである呼び出し側と呼び出される側の *BSW モジュールエントリ(BswModuleEntry)*の *BSW 実行コンテキスト*(*BswExecutionContext*)条件を以下に示す【rte_sws_a_0064】.

表 2-6 直接関数呼び出しの BSW 実行コンテキスト条件

呼び出す側の *BSW 実行コンテキスト*(*BswExecutionContext*)	呼び出される側の *BSW 実行コンテキスト*(*BswExecutionContext*) ○：直接関数呼び出し可能 ×：直接関数呼び出し不可				
	task	interruptCat2	interruptCat1	hook	unspecified
task	○	○	○	×	○
interruptCat2	×	○	○	×	○
interruptCat1	×	×	○	×	○
hook	×	×	×	×	×
unspecified	○	×	×	×	○

BSW スケジューラブルのライフサイクル管理

本 SCHM の機能仕様範囲では，EcuStateManager および OS のライフサイクル管理により，SCHM が行うべき BSW スケジューラブルのライフサイクル管理が実現可能である．このため，本 SCHM では，BSW スケジューラブルのライフサイクル管理を行わない【nrte_sws_0206】.

2.5.5　エクスキュータブル動作の設定

2.5.5.1　ランナブル起動

ランナブル起動の実現方式は，ランナブルを起動する RTE イベント毎に設定する.

ランナブル起動の実行順序

ランナブル起動の実行順番は，ランナブルを起動する RTE イベント毎に設定する.

ランナブル起動の実行順番は，**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**の**ランナブル実行順番(RtePositionInTask)**により指定する【rte_sws_a_0015】.

ランナブル起動の実現方式の設定

ランナブル起動の実現方式として，本 RTEGEN は，以下の選択条件を満たすものを使用する【nrte_sws_0012】.

表 2-7 ランナブル起動の実現方式の設定

実現方式	選択条件
OS タスク起動	**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**が以下の全ての条件を満たす【nrte_sws_0013】. ・ **マッピング先 OS タスク(RteMappedToTaskRef)**が存在する. ・ **使用 OS イベント(RteUsedOsEventRef)**が存在しない.
OS イベント設定	**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**が以下の全ての条件を満たす【nrte_sws_0014】. ・ **マッピング先 OS タスク(RteMappedToTaskRef)**が存在する. ・ **使用 OS イベント(RteUsedOsEventRef)**が存在する.
直接関数起動	**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**が以下の条件を満たす【nrte_sws_0015】. ・ **マッピング先 OS タスク(RteMappedToTaskRef)**が存在しない.

マッピング先 OS タスクの設定

マッピング先 OS タスクは，**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**の**マッピング先 OS タスク(RteMappedToTaskRef)**により指定する【nrte_sws_0016】.

A-RTEGEN は，データ送信を契機とするランナブルを起動するタスクが，以下の OSAP からアクセスできるように設定されていなければならない【nrte_sws_ext_0034】.

・ ECU 間連携の場合
　マスタコアの BSWM 配置パーティションの OSAP
・ ECU 内連携の場合
　データ送信するランナブルが所属するパーティションの OSAP

(※)アクセス権は，OsTaskAccessingApplication に定義する.

使用 OS イベントの設定

使用 OS イベントは，**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**の**使用 OS イベント(RteUsedOsEventRef)**により指定する【nrte_sws_0017】.

マッピング可能な OS オブジェクト

ランナブルは，所属するパーティションの OSAP 以外が管理する OS オブジェクトに割り当てられてはならない【nrte_sws_ext_0016】.

2.5.5.2　BSW スケジューラブル起動

BSW スケジューラブル起動の実現方式は，BSW スケジューラブルを起動する BSW イベント毎に設定する.

BSW スケジューラブル起動の実行順序

BSW スケジューラブル起動の実行順番は，BSW スケジューラブルを起動する BSW イベント毎に設定する．

BSW スケジューラブル起動の実行順番は，**BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**の **BSW スケジューラブル実行順番(RteBswPositionInTask)**により指定する【rte_sws_a_0059】．

BSW スケジューラブル起動の実現方式の設定

BSW スケジューラブル起動の実現方式として，本 RTEGEN は，以下の選択条件を満たすものを使用する【nrte_sws_0207】．

表 2-8 BSW スケジューラブル起動の実現方式の設定

実現方式	選択条件
OS タスク起動	**BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**が以下の全ての条件を満たす【nrte_sws_0208】. ・ **マッピング先 OS タスク(RteBswMappedToTaskRef)**が存在する. ・ **使用 OS イベント(RteBswUsedOsEventRef)**が存在しない.
OS イベント設定	**BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**が以下の全ての条件を満たす【nrte_sws_0209】. ・ **マッピング先 OS タスク(RteBswMappedToTaskRef)**が存在する. ・ **使用 OS イベント(RteBswUsedOsEventRef)**が存在する.
直接関数起動	**BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**が以下の条件を満たす【nrte_sws_0315】. ・ **マッピング先 OS タスク(RteBswMappedToTaskRef)**が存在しない.

マッピング先 OS タスクの設定

マッピング先 OS タスクは，**BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**の**マッピング先 OS タスク(RteBswMappedToTaskRef)**により指定する【nrte_sws_0210】．

使用 OS イベントの設定

使用 OS イベントは，**BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**の**使用 OS イベント(RteBswUsedOsEventRef)**により指定する【nrte_sws_0211】．

マッピング可能な OS オブジェクト

BSW スケジューラブルは，所属するパーティションの OSAP 以外が管理する OS オブジェクトに割り当てられてはならない【nrte_sws_ext_0022】.

2.5.5.3　ランナブルと BSW スケジューラブルの同一タスクへのマッピング

AUTOSAR 仕様では，ランナブルと BSW スケジューラブルは同一タスクにマッピング可能としている【rte_sws_7518】.

サポート範囲の制限

しかし，A-SCHM では，ランナブルと BSW スケジューラブルの同一タスクマッピングはサポートしない【irte_sws_0007】.

2.6　エクスキュータブル周期起動

本節では，ランナブルと BSW スケジューラブルとで共通する仕様はエクスキュータブルとして説明し，仕様差分がある場合のみ個別に説明する．

2.6.1　エクスキュータブル周期起動の構成

エクスキュータブル周期起動は，以下の要素から構成される．要素関係を以下に図示する．

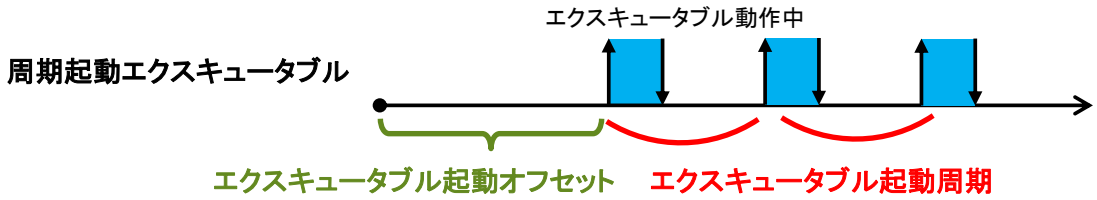

図 2-8 エクスキュータブル周期起動の構成

周期起動エクスキュータブル

周期イベントにより起動し，周期的に動作するエクスキュータブル．

エクスキュータブル起動オフセット

周期起動エクスキュータブルの起動オフセット．

エクスキュータブル起動周期

周期起動エクスキュータブルの起動周期．

2.6.2　エクスキュータブル周期起動の操作

2.6.2.1　周期起動の開始

RTE は，Rte_Start，もしくは Rte_RestartPartition によりパーティションが開始された際に，そのパーティションに所属するランナブルの周期起動を開始する．Rte_Start の呼出し元のコアにマッピングされている周期イベントを起動契機とするランナブルの起動を開始する【rte_sws_7575】.

SCHM は，SchM_Init の呼び出し元のコアにマッピングされている BSW スケジューラブルの周期起動を開始する．SchM_Init の呼出し元のコアにマッピングされている周期イベントを起動契機とする BSW スケジューラブルの起動を開始する【rte_sws_7574】.

2.6.2.2　起動オフセット，および周期の満了

本 RTE/SCHM は，周期イベントで指定した起動オフセット，および起動周期が満たされた際に，周期起動エクスキュータブルを起動する【nrte_sws_0018】.

2.6.3　エクスキュータブル周期起動の実現方式

2.6.3.1　実現方式の構成要素

エクスキュータブル周期起動の実現方式は，以下の要素から構成される．

使用 OS アラーム

周期起動エクスキュータブルの起動契機となる OS アラーム．

OS アラームのセットは，ECU インテグレーション時に実施されることを期待する【nrte_sws_ext_0031】.

使用 OS アラーム起動オフセット

使用 OS アラームの起動オフセット．

使用 OS アラーム起動周期

使用 OS アラームの起動周期．

2.6.3.2　起動オフセット，および周期の満了

RTE/SCHM は，起動オフセット，および周期の満了を OS アラームにより実現する．

RTE/SCHM は，使用 OS アラームからの間接的な満了通知を受け取り，周期起動エクスキュータブルの起動に変換する【rte_sws_a_0016】．間接的な満了通知とは，使用 OS アラームからの OS タスク起動，もしくは OS イベント設定である．

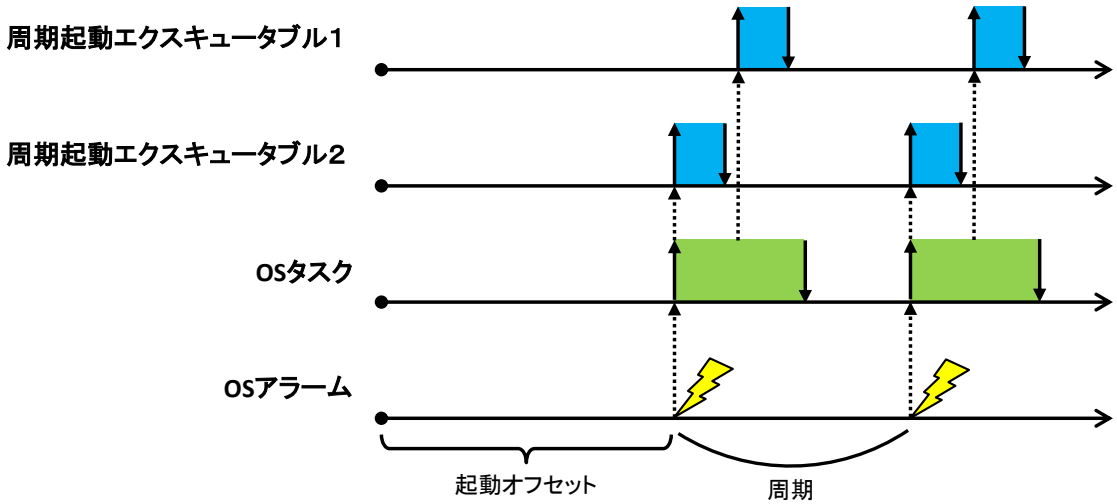

図 2-9 OS アラームによる起動オフセット，および周期の満了の実現

エクスキュータブル-OS アラーム間のタイミング調整

エクスキュータブル起動オフセット／起動周期と使用 OS アラーム起動オフセット／起動周期が異なる場合，RTE/SCHM は，エクスキュータブル起動オフセット／起動周期を満たすように調整を行う【rte_sws_a_0017】．以下に，調整の例を図示する．

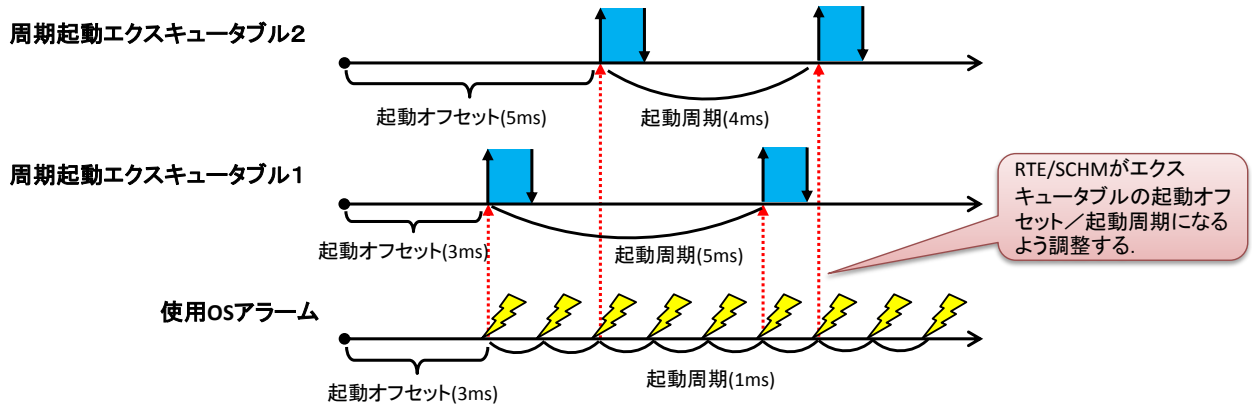

図 2-10 エクスキュータブル-OS アラーム間のタイミング調整

使用上の注意

RTE/SCHM は，周期起動エクスキュータブル間で，動作タイミングを同期する機能を提供しない．RTE/SCHM は，コンフィギュレーションに従った動作を行うのみであり，OS アラームの同期に対する責任は持たない．周期起動のタイミングの同期については，RTE/SCHM，および OS のコンフィギュレーションを行うユーザの責任とする．

ユーザは，以下の方法で周期起動のタイミングの同期を実現できる．

- 全ての周期起動の実現に，同じ OS アラームを使用する．
- 周期起動の実現に，同じ OS カウンタに基づいた異なる OS アラームを使用し，ECU インテグレーションコード中で OS アラームの開始時間の絶対オフセットの同期をとる．
- 周期起動の実現に，同じ設定の異なる OS カウンタ(OS カウンタのインクリメント周期，および最大値が等しい)に基づいた異なる OS アラームを使用する．そして，ECU インテグレーションコード中で OS アラームの開始時間の絶対オフセットの同期をとる．

2.6.4　エクスキュータブル周期起動の設定

2.6.4.1　周期起動エクスキュータブル

周期起動ランナブルは，*周期イベント(TimingEvent)*により起動される*ランナブル(RunnableEntity)*として指定する【nrte_sws_0170】．

周期起動 BSW スケジューラブルは，*周期イベント(BswTimingEvent)*により起動される *BSW スケジ*

ューラブル*(BswSchedulableEntity)*として指定する【nrte_sws_0212】.

2.6.4.2　エクスキュータブル起動オフセット，および起動周期

ランナブル起動オフセット，および起動周期の設定

ランナブル起動オフセットは，*周期イベント(TimingEvent)*を参照する **RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**の**起動オフセット(RteActivationOffset)**により指定する【rte_sws_7000】.

ランナブル起動周期は，*周期イベント(TimingEvent)*の*起動周期(period)*により指定する【rte_sws_a_0019】.

BSW スケジューラブル起動オフセット，および起動周期の設定

BSW スケジューラブル起動オフセットは，*周期イベント(BswTimingEvent)*を参照する **BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**の**起動オフセット(RteBswActivationOffset)**により指定する【rte_sws_7520】.

BSW スケジューラブル起動周期は，*周期イベント(BswTimingEvent)*の*起動周期(period)*により指定する【rte_sws_a_0060】.

2.6.4.3　使用 OS アラーム

ランナブルの使用 OS アラームは，*周期イベント(TimingEvent)*を参照する **RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**の**使用 OS アラーム(RteUsedOsAlarmRef)**により指定する【rte_sws_7804】【rte_sws_7806】.

BSW スケジューラブルの使用 OS アラームは，*周期イベント(BswTimingEvent)*を参照する **BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**の**使用 OS アラーム(RteBswUsedOsAlarmRef)**により指定する【nrte_sws_0213】.

使用 OS アラーム起動オフセットは，使用 OS アラームを参照する **OS アクティベーション設定(RteUsedOsActivation)**の**起動オフセット期待値(RteExpectedActivationOffset)**により指定する【rte_sws_7805】.

使用 OS アラーム起動周期は，使用 OS アラームを参照する **OS アクティベーション設定(RteUsedOsActivation)**の**起動周期期待値(RteExpectedTickDuration)**により指定する【nrte_sws_0019】.

使用上の注意

エクスキュータブル周期起動を OS アラームによって実現するため，周期起動エクスキュータブルの最悪実行時間は，そのエクスキュータブル周期起動を実現するための使用 OS アラーム起動周期を下回っていなければならない【nrte_sws_ext_0002】．以下に問題が起きる場合を図示する．本仕様に違反

しないよう，SW-C/BSWM を開発するユーザがエクスキュータブルの処理内容を設計／実装するか，RTE/SCHM のコンフィギュレーションを行うユーザがエクスキュータブル周期起動の設定を行う必要がある．

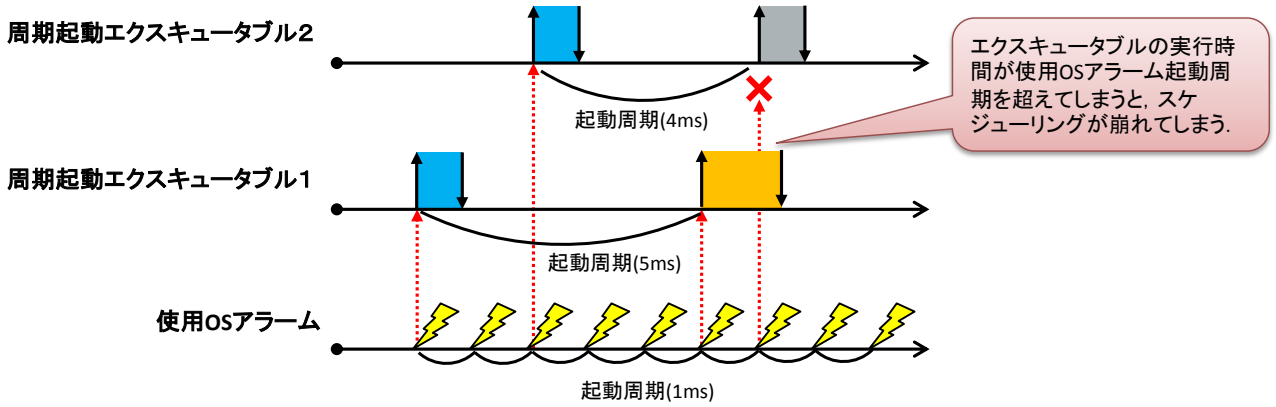

図 2-11 使用 OS アラーム起動周期を周期起動エクスキュータブルの実行時間が超える場合

2.7　エクスキュータブルバックグラウンド起動

本節では，ランナブルと BSW スケジューラブルとで共通する仕様はエクスキュータブルとして説明し，仕様差分がある場合のみ個別に説明する．

2.7.1　エクスキュータブルバックグラウンド起動の構成

エクスキュータブルバックグラウンド起動は，以下の要素から構成される．要素関係を以下に図示する．

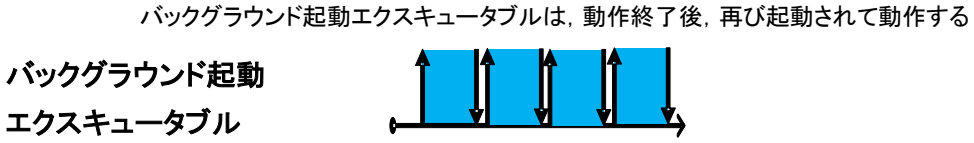

図 2-12 エクスキュータブルバックグラウンド起動の構成

バックグラウンド起動エクスキュータブル

バックグラウンドイベントにより起動し，動作するエクスキュータブル．

2.7.2　エクスキュータブルバックグラウンド起動の操作

2.7.2.1　起動の開始

RTE は，Rte_Start の呼出しにより，バックグラウンドイベントを起動契機とするランナブルの起動を開始する【rte_sws_7178】．

SCHM は，SchM_Init の呼び出しにより，バックグラウンドイベントを起動契機とする BSW スケジューラブルの起動を開始する【rte_sws_7584】．

本 RTE/SCHM は，バックグラウンドイベントを起動契機とするエクスキュータブルが終了した際，エクスキュータブルを再度起動する【nrte_sws_0358】．

2.7.3　エクスキュータブルバックグラウンド起動の実現方式

バックグラウンド起動の実現方式は，以下のいずれかである．どの方式によりエクスキュータブルを起動するかは，コンフィギュレーションにより選択できる．

表 2-9 バックグラウンド起動の実現方式

バックグラウンド起動の実現方式	説明
周期不定	最低優先度の走り続けるタスクによって，エクスキュータブルを起動し続ける．そのため，スケジューリングポリシがノンプリエンプティブスケジューリングのタスクで起動する場合には注意が必要となる． コア内の最低優先度のタスクにマッピングするため，複数パーティションにマッピングする場合は利用できない．
周期固定	OS アラームからの満了通知によって，エクスキュータブルを周期的に起動する． 周期不定の複数パーティション制限を回避する場合や，CPU 資源を効率的に利用したい場合に利用できる．

表 2-10 バックグラウンド起動とランナブル起動の実現方式の対応表

バックグラウンド起動の実現方式	ランナブル起動の実現方式		
	OS タスク起動	OS イベント設定	直接関数起動
周期不定	○	×	×
周期固定	○	○	×

表 2-11 バックグラウンド起動と BSW スケジューラブル起動の実現方式の対応表

バックグラウンド起動の実現方式	BSW スケジューラブル起動の実現方式		
	OS タスク起動	OS イベント設定	直接関数起動
周期不定	○	×	×
周期固定	○	○	×

2.7.3.1　実現方式の構成要素

エクスキュータブルバックグラウンド起動の実現方式は，以下の要素から構成される．

使用 OS アラーム

バックグラウンド起動エクスキュータブルの起動契機となる OS アラーム．

OS アラームのセットは，ECU インテグレーション時に実施されることを期待する〔nrte_sws_ext_0031〕．

2.7.3.2　周期不定起動

本 RTE/SCHM は，バックグラウンドイベントを起動契機とするエクスキュータブルを周期不定で起動する場合，エクスキュータブルの起動/再起動をタスクボディ内の無限ループにより実現する【nrte_sws_0359】.

RTE/SCHM は，OS タスクの起動により，バックグラウンド起動エクスキュータブルを起動する．OS タスクにマッピングされたバックグラウンド起動エクスキュータブルは，直前のエクスキュータブルの終了に伴い順次起動される．全ての実行が終了すると，再び全てのエクスキュータブルを順次起動する．

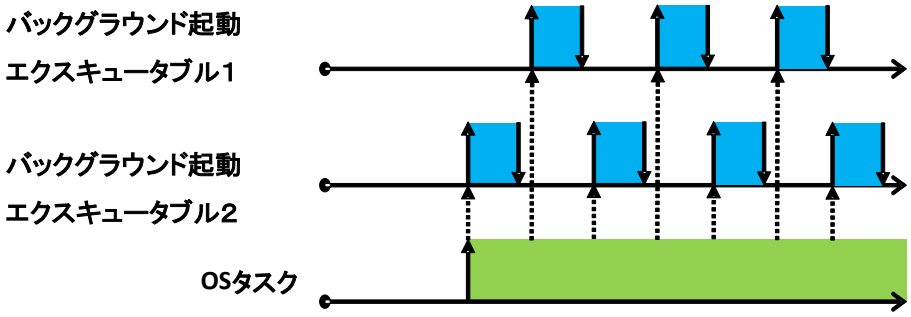

図 2-13 周期不定起動によるバックグラウンド起動エクスキュータブル起動

2.7.3.3　周期固定起動

本 RTE/SCHM は，バックグラウンドイベントを起動契機とするエクスキュータブルを周期固定で起動する場合，エクスキュータブルの起動/再起動を OS アラームの満了通知により実現する【nrte_sws_0360】.

RTE/SCHM は，使用 OS アラームからの間接的な満了通知を受け取り，バックグラウンド起動エクスキュータブルを起動する．間接的な満了通知とは，使用 OS アラームからの OS タスク起動，もしくは OS イベント設定である．

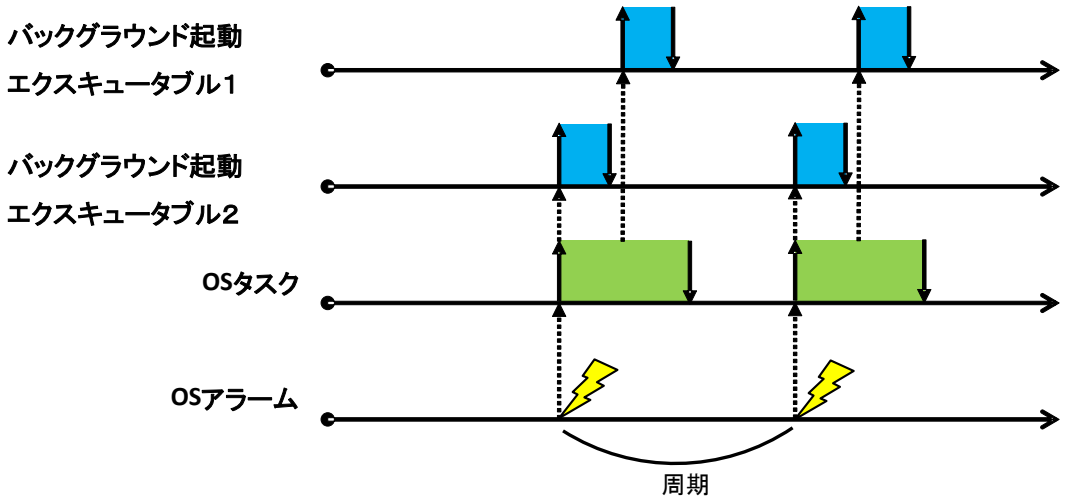

図 2-14 周期固定起動によるバックグラウンド起動エクスキュータブル起動

使用上の注意

RTE/SCHM は，OS アラームの満了通知に従った動作を行うのみであり，バックグラウンド起動のタイミングについては，OS のコンフィギュレーションを行うユーザの責任とする．

2.7.4　エクスキュータブルバックグラウンド起動の設定

2.7.4.1　バックグラウンド起動エクスキュータブル

バックグラウンド起動ランナブルは，*バックグラウンドイベント(BackgroundEvent)*により起動される *ランナブル(RunnableEntity)*として指定する【nrte_sws_0361】．

バックグラウンド起動 BSW スケジューラブルは，*バックグラウンドイベント(BswBackgroundEvent)*により起動される *BSW スケジューラブル(BswSchedulableEntity)*として指定する【nrte_sws_0362】．

2.7.4.2　使用 OS アラーム

ランナブルの使用 OS アラームは，*バックグラウンドイベント(BackgroundEvent)*を参照する **RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**の**使用 OS アラーム(RteUsedOsAlarmRef)**により指定する【rte_sws_7179】．

表 2-12 バックグラウンド起動ランナブルの設定

バックグラウンド起動の実現方式	選択条件
周期不定	**使用 OS アラーム(RteUsedOsAlarmRef)**が指定されない場合，バックグラウンド起動エクスキュータブルは，周期不定起動により動作する【nrte_sws_0363】.
周期固定	**使用 OS アラーム(RteUsedOsAlarmRef)**が指定された場合，バックグラウンド起動エクスキュータブルは，周期固定起動により動作する【nrte_sws_0364】.

BSW スケジューラブルの使用 OS アラームは，*バックグラウンドイベント(BswBackgroundEvent)*を参照する **BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**の**使用 OS アラーム(RteBswUsedOsAlarmRef)**により指定する【rte_sws_a_0067】.

表 2-13 バックグラウンド起動 BSW スケジューラブルの設定

バックグラウンド起動の実現方式	選択条件
周期不定	**使用 OS アラーム(RteBswUsedOsAlarmRef)**が指定されない場合，バックグラウンド起動エクスキュータブルは，周期不定起動により動作する【nrte_sws_0365】.
周期固定	**使用 OS アラーム(RteBswUsedOsAlarmRef)**が指定された場合，バックグラウンド起動エクスキュータブルは，周期固定起動により動作する【nrte_sws_0366】.

2.8　S/R 連携

RTE は，データ要素の送受信機能(S/R 連携)を提供する．このデータ要素は，１つの SW-C から送信され，１つ以上の SW-C で受信される．S/R 連携は単方向の連携であるため，受信側 SW-C から応答を返す場合には，別の S/R 連携として応答を送信する必要がある．

2.8.1　S/R 連携の構成

S/R 連携は以下の要素から構成される．データの値などの設定値以外の要素関係を以下に図示する．

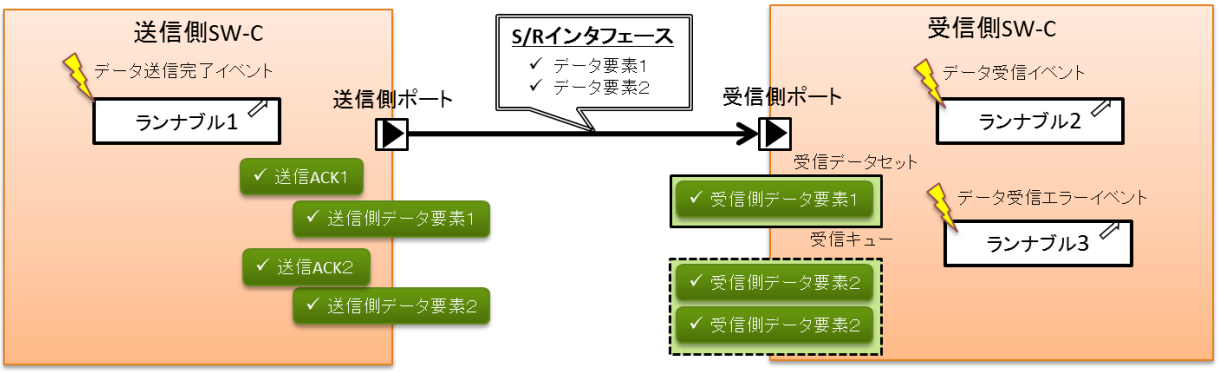

図 2-15 S/R 連携の構成

S/R インタフェース

S/R 連携におけるポートインタフェースであり，SW-C 間のデータ送受信内容を規定するインタフェース仕様を表す．S/R インタフェースには，複数のデータ要素を含むことができる．

データ要素

S/R 連携のインタフェース要素であり，S/R 連携で実際にやりとりされるデータ内容を規定する．

送信側ポート

S/R 連携における提供側ポートであり，データを送信する SW-C のポート．

受信側ポート

S/R 連携における要求側ポートであり，データを受信する SW-C のポート．

送信側データ要素

送信側ポートにおけるデータ要素のインスタンス．データ送信，送信 ACK 取得のそれぞれの API はこの単位で生成される．

受信側データ要素

受信側ポートにおけるデータ要素のインスタンス．受信データ取得のための API はこの単位で生成される．

送信側 SW-C

S/R 連携における提供側 SW-C であり，S/R インタフェースを介してデータを送信する SW-C.

受信側 SW-C

S/R 連携における要求側 SW-C であり，S/R インタフェースを介してデータを受信する SW-C.

受信データセット

データセマンティクスにおける受信した値，および状態を保持するためのデータ領域．受信側データ要素毎に存在する．データセマンティクスについては，2.8.2.3 節を参照．

受信キュー

イベントセマンティクスにおける受信した値を保持するためのキュー．受信側データ要素毎に存在する．イベントセマンティクスについては，2.8.2.3 節を参照．

初期値

受信側 SW-C がデータを受信する前に，データを取得しようとした際に返される値【rte_sws_6009】．受信側データ要素毎に設定可能である．

無効値

データが現在無効であると識別するための値．送信側データ要素毎に設定可能である．

タイムアウト時間

データがタイムアウトしたと判定するまでの時間．受信側データ要素毎に設定可能である．

送信 ACK

データ要素に対する送信 ACK．送信側データ要素毎に設定可能である．

データ送信完了イベント

データの送信成功/失敗時にランナブルを起動するイベント．

データ受信イベント

データの受信時にランナブルを起動するイベント．

データ受信エラーイベント

データの受信におけるエラー発生時にランナブルを起動するイベント.

2.8.2　S/R 連携の種別

2.8.2.1　S/R 連携の多重度

RTE は，以下の多重度の S/R 連携をサポートする．連携の多重度については，2.4.2.2 節を参照.

- 1:0 連携，0:1 連携〔rte_sws_1329〕
- 1:1 連携
- 1:N 連携
- N:1 連携
- N:M 連携

以上のいずれのケースについても，RTE は，送信側ポートに対し 1 つの送信用 API，受信側ポートに対し 1 つの受信用 API を提供する.

送信側 SW-C は受信側ポートの数に関わらず，値を送信するために提供された 1 つの API を呼び出すのみでよい．同様に，受信側 SW-C は受信したデータを取得するために 1 つの API を呼び出すのみでよい.

複数の受信側ポートへの送信の同期性

RTE は，複数の受信側ポートに対してデータを送信する場合に，伝搬されるデータが複数の受信側ポートに同時に届き，参照可能になることを保証しない【nrte_sws_0059】.

2.8.2.2　データ要素数

S/R インタフェースは，1 つ以上のデータ要素を含む.

RTE は，データ要素毎に異なるデータ送受信のための API を提供し，データ要素毎に独立して送受信を行う【rte_sws_6008】.

2.8.2.3　データのバッファリング

データのバッファリングは受信側ポートで行う．受信側ポートでは，以下のいずれかのバッファリングを行う．いずれの方法でバッファリングを行うかはコンフィギュレーションにより選択できる.

バッファリング方法	内容
データセマンティクス	データセマンティクスは，最新の値のみを保持するバッファリング方式である.
イベントセマンティクス	イベントセマンティクスは，イベント(受信した値)の損失を避けるため，受信したイベントを保持するためのキューイングを行うバッファリング方式である.

データセマンティクスによるバッファリング

RTEGEN は，受信ポートのデータセマンティクスのデータ要素毎に，キューではない単一のデータセット(受信データセット)を実装する【rte_sws_2516】.

〔rte_sws_7046〕で規定される初期化条件を満たす場合，RTE は，受信データセットの値を初期値で初期化する【rte_sws_2517】.

イベントセマンティクスによるバッファリング

RTEGEN は，受信ポートのイベントセマンティクスのデータ要素毎に，受信キューを実装する【rte_sws_2521】．受信キューは，FIFO で実現する【rte_sws_2522】.

2.8.2.4　データの無効化

RTE は，データ要素を送信側 SW-C で無効化する機能を提供する．送信側 SW-C は，受信側 SW-C に対してデータが無効となったことを通知することができる．RTE が本機能を提供するかは，コンフィギュレーションにより選択できる.

RTE は，データ要素の無効化を，データセマンティクスのデータ要素に対してのみサポートする【rte_sws_5033】.

無効値受信時処理

受信側ポートがデータ無効化の通知を受け取った際の処理方法は以下のいずれかの方法をとる．いずれの方法により無効化処理を行うかは，コンフィギュレーションにより選択できる.

無効値受信時処理	内容
keep	受信側ポートが無効値のデータを受信した場合，無効値の受信データをそのまま保持し，受信データセットの状態を「無効状態」に移行する.
replace	受信側ポートが無効値のデータを受信した場合，受信データを初期値にリセットし，受信データセットの状態を「正常状態」に移行する.

2.8.2.5　送信 ACK

RTE は，データが正しく送信されたかを確認する送信 ACK 機能を提供する．送信側 SW-C は，本機能を使用することで送信が正しく行われたかを確認することができる．なお，本機能により確認できるのはデータが正しく送信されたかのみであり，受信側 SW-C に正しく届いたかは確認できないことに注意が必要である．RTE が本機能を提供するかは，コンフィギュレーションにより選択できる.

RTE は，データ要素毎に独立した送信 ACK をサポートする【rte_sws_5504】.

2.8.2.6　データのタイムアウト監視

S/R 連携が物理ネットワークを経由する場合，通信が失敗し，受信側でデータ更新が一定時間，もしくは，永遠に行われない可能性がある．RTE はこのような状況を監視し，タイムアウトを通知する機能を提供する．RTE が本機能を提供するかは，コンフィギュレーションにより選択できる.

2.8.2.7　データのフィルタ

RTE は，データの値に依存するフィルタ機能(一定の条件を満たす値で受信されたデータだけが，受信側 SW-C に到達できる)を提供する．フィルタはデータセマンティクスの受信側データ要素に設定する．RTE が本機能を提供するかは，コンフィギュレーションにより選択できる．

S/R インタフェース内のデータ要素毎に，異なるフィルタを指定可能としなければならない【rte_sws_5501】．

フィルタの種別

RTE は，以下のフィルタをサポートする．フィルタのアルゴリズムは，関連文書「OSEK/VDX Communication」の規定に準拠する．フィルタのアルゴリズムの詳細は，関連文書「OSEK/VDX Communication」を参照.

- always
- maskedNewDiffersMaskedOld
- maskedNewDiffersX
- maskedNewEqualsX
- never
- newIsOutside
- newIsWithin
- oneEveryN

2.8.2.8　受信モード

ランナブルの起動

受信するランナブルは，新しいデータが利用可能になったときに，RTE によって起動される．

受信モード「ランナブルの起動」はカテゴリ 1A，1B，もしくは 2 のランナブルに対して有効である【rte_sws_6007】.

2.8.3　S/R 連携の状態

2.8.3.1　送信 ACK の状態

送信 ACK のステータスの状態遷移を以下に定義する【nrte_sws_0376】.

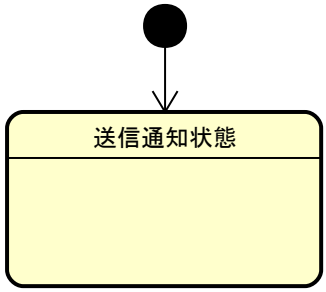

図 2-16 ECU 内連携における受信データセットの状態遷移図

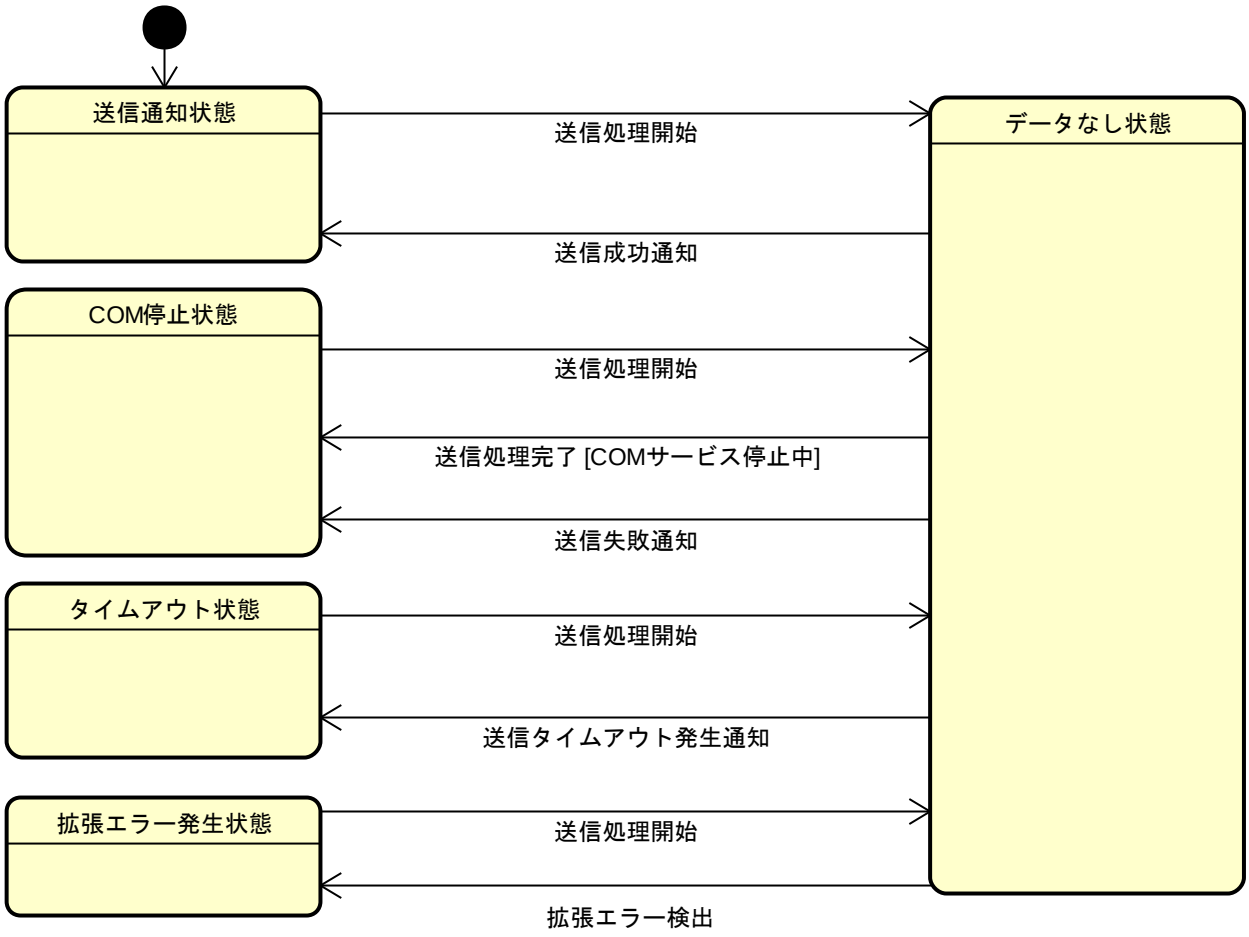

図 2-17 ECU 間連携における受信データセットの状態遷移図

表 2-14 送信 ACK の状態

送信 ACK の状態	説明
送信通知状態	送信 ACK を受信した.
データなし状態	送信 ACK，もしくは送信失敗通知を COM から受信していない.
COM 停止状態	COM サービス停止により送信に失敗した．あるいは，COM からタイムアウト発生通知を受信する前に送信失敗通知を受信した.
タイムアウト状態	COM から送信失敗通知を受信する前にタイムアウト発生通知を受信した.
拡張エラー発生状態	A-RTE がデータ伝搬で使用する OS/IOC/COM のシステムサービスにおいて，拡張エラーが発生した.

表 2-15 送信 ACK の遷移

送信 ACK の遷移	説明
送信処理開始	データ送信／データ無効化処理が開始された.
送信処理完了[COM サービス停止中]	COM サービス停止中に送信処理が完了した.
送信成功通知	COM から送信成功通知を受信した.
送信失敗通知	COM から送信失敗通知を受信した.
送信タイムアウト発生通知	COM からタイムアウト発生通知を受信した.
拡張エラー検出	下記の拡張エラーを検出した. ・ COM API が COM サービス停止中以外のエラーを返す ・ マスタコアの非信頼パーティションからの ECU 間データ送信において，OS 信頼関数の呼び出しがエラーを返す ・ スレーブコアからの ECU 間データ送信において，データのコア間送信の IOC 呼び出しがエラーを返す

表 2-16 ECU 間における送信 ACK の状態遷移表

遷移 送信 ACK の状態	送信処理開始	送信処理完了[COM サービス停止中]	送信成功通知	送信失敗通知	送信タイムアウト発生通知	拡張エラー検出
送信通知状態	データなし状態	―	―	―	―	―
データなし状態	―	COM 停止状態	送信通知状態	COM 停止状態	タイムアウト状態	拡張エラー発生状態
COM 停止状態	データなし状態	―	―	―	―	―
タイムアウト状態	データなし状態	―	―	―	―	―
拡張エラー発生状態	データなし状態	―	―	―	―	―

2.8.3.2　受信データセットの状態

データセマンティクスの受信データセットの状態遷移を以下に定義する.

本 RTE では，受信データセットのデータが無効化されており，かつ，タイムアウト時間が経過した場合を「無効化／タイムアウト状態」と定義し，無効化／タイムアウト状態では「無効化状態」および「タイムアウト状態」の両状態における受信データセットの挙動を示すものとする【nrte_sws_0061】.
データ受信とデータ無効化により，タイムアウト計測はリスタートされる【rte_sws_8004】.

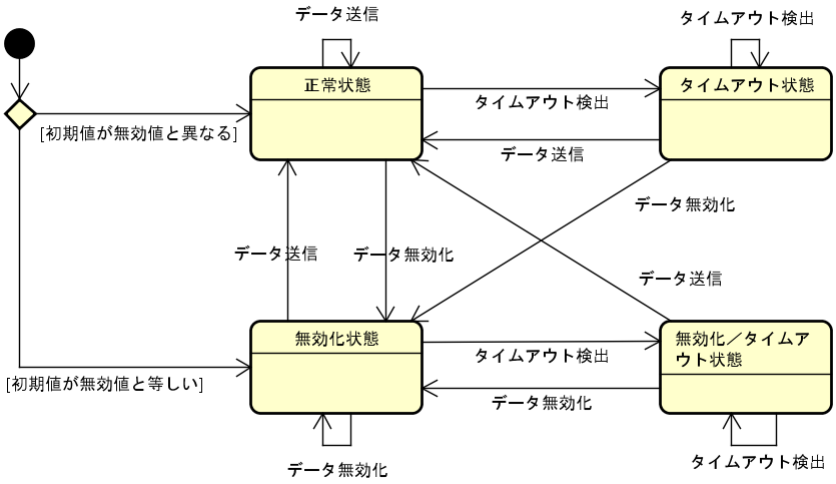

図 2-18 無効値受信時処理 keep における受信データセットの状態遷移図

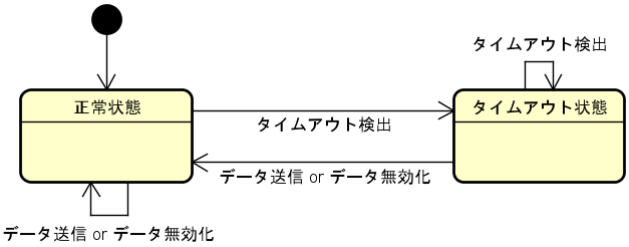

図 2-19 無効値受信時処理 replace における受信データセットの状態遷移図

表 2-17 受信データセットの状態

受信データセットの状態	説明
正常状態	データが正常に受信できた．あるいは，まだ一度もデータを受信していない．
無効化状態	データが無効化された． 無効値受信時処理が keep であり，データセマンティクスのデータ要素の初期値が無効値と等しい場合，RTE は，受信データセットの初期状態を「無効化状態」とする【rte_sws_8008】.
タイムアウト状態	前回データを受信してからコンフィギュレーションで指定したタイムアウト時間が経過した．
無効化／タイムアウト状態	データが無効化された，かつ，前回データを受信してからコンフィギュレーションで指定したタイムアウト時間が経過した．

表 2-18 受信データセットの遷移

受信データセットの遷移	説明（箇条書きはアクションの内容）
データ送信	送信側ポートからデータが送信された． ・　2.8.4.1 データセマンティクスのデータ送信の挙動に示す処理を行う．
データ無効化	送信側ポートから無効値が送信された． ・　2.8.4.2 データ無効化の挙動に示す処理を行う．
タイムアウト検出	一定時間のデータ受信がないこと，あるいはパーティションの停止／再起動によりタイムアウトが通知された． ・　2.8.4.4 タイムアウトの挙動に示す処理を行う．

表 2-19 無効値受信時処理 keep における受信データセットの状態遷移表

遷移 受信データセットの状態	データ送信	データ無効化	タイムアウト検出
正常状態	正常状態	無効化状態	タイムアウト状態
無効化状態	正常状態	無効化状態	無効化／タイムアウト状態
タイムアウト状態	正常状態	無効化状態	タイムアウト状態
無効化／タイムアウト状態	正常状態	無効化状態	無効化／タイムアウト状態

以上の表にて，データセマンティクスのデータ要素の初期値が無効値と等しい場合，「無効化状態」が初期状態となる．そうでない場合，正常状態が初期状態となる(表 2-17).

表 2-20 無効値受信時処理 replace における受信データセットの状態遷移表

遷移 受信データセットの状態	データ送信	データ無効化	タイムアウト検出
正常状態（初期状態）	正常状態	正常状態	タイムアウト状態
タイムアウト状態	正常状態	正常状態	タイムアウト状態

2.8.3.3　受信キューの状態

イベントセマンティクスの受信キューの状態遷移を以下に定義する．

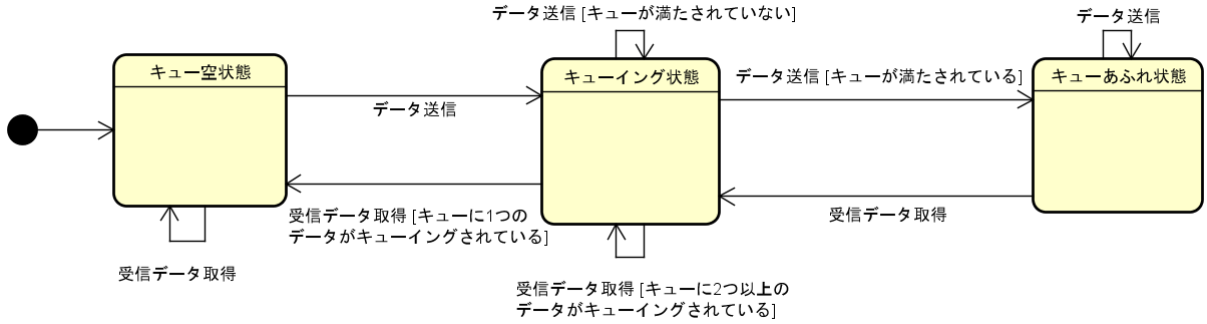

図 2-20 イベントセマンティクスの受信キューの状態遷移図

表 2-21 受信キューの状態

受信キューの状態	説明
キュー空状態	キューのデータが全て取り出された．あるいは，まだデータを受信していない．
キューイング状態	データが正常に受信でき，キューイングされた．
キューあふれ状態	キューが満たされているためデータが受信できず破棄された．

表 2-22 受信キューの遷移

受信キューの遷移	説明
データ送信	送信側ポートからデータが送信された. ・ 2.8.4.1 イベントセマンティクスのデータ送信の挙動に示す処理を行う.
受信データ取得	受信側ポートでデータを取得した. ・ 2.8.4.3 イベントセマンティクスの受信データ取得の挙動に示す処理を行う.

表 2-23 受信キューの状態遷移表

遷移 受信キューの状態	データ送信	受信データ取得
キュー空状態（初期状態）	キューイング状態	キュー空状態
キューイング状態	[キューが満たされていない] キューイング状態 [キューが満たされている] キューあふれ状態	[キューに 1 つのデータがキューイングされている] キュー空状態 [キューに 2 つ以上のデータがキューイングされている] キューイング状態
キューあふれ状態	キューあふれ状態	キューイング状態

2.8.4　S/R 連携の操作

S/R 連携の操作の一貫性保証

RTE は，S/R 連携におけるデータ要素の送受信がアトミックに行われることを保証する【rte_sws_4527】.

2.8.4.1　データ送信

RTE は，データ要素毎に，データ送信のための API(データセマンティクスの場合，Rte_Write，イベントセマンティクスの場合，Rte_Send)を提供する【rte_sws_6011】． 送信側 SW-C は，提供された API を使用することができる.

データセマンティクスのデータ送信の挙動

Rte_Write によりデータ送信が開始された場合の挙動は以下の通りである【nrte_sws_0067】．RTE は，各受信側ポートへの値の伝搬を独立して行うため，(4)以降の手順を受信側ポート毎に並行して実施する.

(1) 送信 ACK のステータスを「送信 ACK 応答なし」に移行する.
(2) データ送信時に指定された値を受信側ポートに送信する.
(3) 送信が成功，もしくは失敗した時点で，送信 ACK のステータスを送信の結果に応じた状態に移行し，データ送信完了イベントを起動契機とするランナブルを起動する.
(4) データ送信時に指定された値を受信側ポートに伝搬する.
(5) 受信側ポートにおいてフィルタが有効である場合，規定のアルゴリズムに従い値のフィルタを実施する．フィルタ対象となった場合，(6)以降を実施しない.
(6) 受信側ポートにおいて伝搬が完了すると，受信データセットを伝搬された値に設定する【rte_sws_2520】．そして，受信データセットの状態を「正常状態」に移行する.
(7) データ受信イベントを起動契機とするランナブルを起動する.

イベントセマンティクスのデータ送信の挙動

Rte_Send によりデータ送信が開始された場合の挙動は以下の通りである【nrte_sws_0068】．RTE は，各受信側ポートへの値の伝搬を独立して行うため，(4)以降の手順を受信側ポート毎に並行して実施する.

(1) 送信 ACK のステータスを「送信 ACK 応答なし」に移行する.
(2) データ送信時に指定された値を受信側ポートに送信する.
(3) 送信が成功，もしくは失敗した時点で，送信 ACK のステータスを送信の結果に応じた状態に移行し，データ送信完了イベントを起動契機とするランナブルを起動する.
(4) データ送信時に指定された値を受信側ポートに伝搬する.
(5) 受信側ポートにおいて伝搬が完了すると，以下の方法で受信への値のキューイングを行う.
キューが満たされていない場合，値をキューの最後に追加し，受信キューの状態を「キューイング状態」に移行する.
キューが満たされている場合，値を破棄し，受信キューの状態を「キューあふれ状態」に移行する【rte_sws_2523】.
(6) データ受信イベントを起動契機とするランナブルを起動する.

データ送信時に検出したエラーの影響

ある受信側ポートへの値の送信中に検出したエラーが，他の受信側ポートへの値の伝搬に影響を与えないことを保証する【rte_sws_1082】.

受信側ポートへの値の伝搬順序

複数の受信側ポートが存在する場合，どの受信側ポートから順に値の伝搬を開始するかの順序は実装定義とする【nrte_sws_0020】.

受信側ポートへの値の伝搬の同期性

RTE は，全受信側ポートへ値が同時に伝搬されることを保証しない【rte_sws_a_0020】．また受信

側ポートへ値が伝搬される順番についても保証しない【nrte_sws_0021】.

<u>複数のランナブルを起動する場合の実行順番</u>

データ送信において起動する複数の異なるイベントが存在する場合，ランナブルの起動順番は以下とする【nrte_sws_0373】.

起動順番	起動対象	詳細条件
1	OS タスクにマッピングされているエクスキュータブル (OS タスク単位で起動する)	OS タスク内のエクスキュータブルの起動順番は，〔rte_sws_a_0059〕に従う．OS タスク間の起動順番は以下とする． 起動順番 / 詳細条件 1 / OS タスク優先度が高い順 2 / 同じ優先度の OS タスクが存在する場合は，OS タスクの名前順(アルファベットの昇順) 3 / 同じ優先度の OS タスクが存在し，かつ，イベント設定の場合は，OS イベントの名前順(アルファベットの昇順)

2.8.4.2　データ無効化

RTE は，データ要素毎に，データ無効化のための API(Rte_Invalidate)を提供する．送信側 SW-C は，提供された API を使用することができる.

<u>データ無効化の挙動（無効値受信時処理 keep の場合）</u>

無効値受信時処理 keep の場合に，Rte_Invalidate によりデータ無効化が開始された際の挙動は以下の通りである【nrte_sws_0069】．RTE は，各受信側ポートへの無効値の伝搬を独立して行うため，(4)以降の手順を受信側ポート毎に並行して実施する.

(1) 送信 ACK のステータスを「送信 ACK 応答なし」に移行する.
(2) データ送信時に指定された値を受信側ポートに送信する.
(3) 送信が成功，もしくは失敗した時点で，送信 ACK のステータスを送信の結果に応じた状態に移行し，データ送信完了イベントを起動契機とするランナブルを起動する.
(4) 送信側ポートの無効値を受信側ポートに伝搬する.
(5) 受信側ポートにおいて伝搬が完了すると，受信データセットを無効値に設定し，受信データセットの状態を「無効化状態」に移行する.
(6) データ受信エラーイベントを起動契機とするランナブルを起動する.

無効値受信時処理 keep の場合，データのフィルタ処理は実施しない【nrte_sws_0070】.

<u>データ無効化の挙動（無効値受信時処理 replace の場合）</u>

無効値受信時処理 replace の場合に，Rte_Invalidate によりデータ無効化が開始された際の挙動は以下の通りである【nrte_sws_0071】．RTE は，各受信側ポートへの無効値の伝搬を独立して行うため，(4)以降の手順を受信側ポート毎に並行して実施する．

(1) 送信 ACK のステータスを「送信 ACK 応答なし」に移行する．
(2) データ送信時に指定された値を受信側ポートに送信する．
(3) 送信が成功，もしくは失敗した時点で，送信 ACK のステータスを送信の結果に応じた状態に移行し，データ送信完了イベントを起動契機とするランナブルを起動する．
(4) 送信側ポートの無効値を受信側ポートに伝搬する．送信側ポートの無効値を受信側ポートに伝搬する．
(5) 受信側ポートで伝搬が開始すると，伝搬される値を初期値に置き換える．
(6) 受信側ポートにおいてフィルタが有効である場合，規定のアルゴリズムに従い値のフィルタを実施する．フィルタ対象となった場合，(4)以降を実施しない．
(7) 受信側ポートにおいて置き換え後の値を受信データセットに設定し，その受信データセットの状態を「正常状態」に移行する．
(8) データ受信イベントを起動契機とするランナブルを起動する．

<u>受信側ポートへの無効値の伝搬順序</u>

複数の受信側ポートが存在する場合，どの受信側ポートから順に値の伝搬を開始するかの順序は実装定義とする【nrte_sws_0022】．

<u>受信側ポートへの無効値の伝搬の同期性</u>

RTE は，全受信側ポートへ無効値が同時に伝搬されることを保証しない【rte_sws_a_0021】．また受信側ポートへ無効値が伝搬される順番についても保証しない【nrte_sws_0023】．

<u>複数のランナブルを起動する場合の実行順番</u>

データ無効化において起動する複数の異なるイベントが存在する場合，ランナブルの起動順番は以下とする【nrte_sws_0374】.

<table>
<tr><th>起動順番</th><th>起動対象</th><th>詳細条件</th></tr>
<tr><td>1</td><td>OS タスクにマッピングされているエクスキュータブル
(OS タスク単位で起動する)</td><td>OS タスク内のエクスキュータブルの起動順番は，〔rte_sws_a_0059〕に従う．OS タスク間の起動順番は以下とする．
<table>
<tr><th>起動順番</th><th>詳細条件</th></tr>
<tr><td>1</td><td>OS タスク優先度が高い順</td></tr>
<tr><td>2</td><td>同じ優先度の OS タスクが存在する場合は，OS タスクの名前順(アルファベットの昇順)</td></tr>
<tr><td>3</td><td>同じ優先度の OS タスクが存在し，かつ，イベント設定の場合は，OS イベントの名前順(アルファベットの昇順)</td></tr>
</table>
</td></tr>
</table>

2.8.4.3　受信データ取得

RTE は，データ要素毎に，受信データ取得のための API(データセマンティクスの場合，Rte_Read，イベントセマンティクスの場合，Rte_Receive)を提供する【rte_sws_6011】．　受信側 SW-C は，提供された API を使用することができる．

データセマンティクスの受信データ取得の挙動

Rte_Read により受信データ取得が開始された場合，受信データセットの値を API の OUT 引数に設定する【rte_sws_2518】．受信側データセットの状態を API の返り値として受信側 SW-C に返す〔rte_sws_1093〕〔rte_sws_2626〕〔rte_sws_2703〕〔rte_sws_7690〕．

イベントセマンティクスの受信データ取得の挙動

Rte_Receive により受信データ取得が開始された場合の挙動は以下の通りである【nrte_sws_0073】.

(1) 受信キューの状態が「キューイング状態」の場合，受信キューから値を取り出し，取り出した値を OUT 引数に設定する．受信キューの状態を API の返り値として返す.

(2) 受信キューの状態が「キュー空状態」の場合，受信キューの状態を API の返り値として返す【rte_sws_2525】.

(3) 受信キューの状態が「キューあふれ状態」の場合，受信キューから値を取り出し，取り出した値を OUT 引数に設定する．受信キューの状態を「キューイング状態」に移行する【rte_sws_2524】.「キューあふれ状態」を API の返り値として返す.

2.8.4.4　送信 ACK

送信 ACK が有効である場合，RTE は SW-C に対し，送信の応答，もしくはエラー発生を通知する

【rte_sws_1080】．*送信 ACK 要求(TransmissionAcknowledgementRequest)*が指定された場合，RTE は送信 ACK の受信モードに関わらずタイムアウト監視の実行を保証する【rte_sws_3754】.

データ要素の送信毎に，RTE は 1 回の送信 ACK(送信の成功，もしくは失敗)を送信側の SW-C に渡す【rte_sws_3756】．送信の成功，もしくは失敗についてのステータス情報は，送信 ACK を取得するための RTE API の返り値として取得する【rte_sws_3757】．送信の成功，もしくは失敗についてのステータス情報は，last-is-best セマンティクスでバッファリングされ，データ要素の送信毎にステータス情報はリセットされる【rte_sws_3604】.

提供側データ要素が以下の条件を満たす場合，送信 ACK が発生する，もしくは RTE によりタイムアウトが検出された際に，*データ送信完了イベント(DataSendCompletedEvent)*により参照されるランナブルを起動する【rte_sws_1286】.

・送信 ACK が有効である．
・提供側データ要素にひもづくデータ送信完了イベントが，いずれかのランナブルを参照している．

<u>パーティション内連携におけるイベント発火</u>

パーティション内連携の場合，データ送信完了イベントは以下の条件を満たす場合のみ発火する【rte_sws_8018】.

- 全ての受信者に対する送信を実施する．

<u>パーティション内連携におけるイベント発火</u>

パーティション間連携の場合，データ送信完了イベントは以下の全ての条件を満たす場合のみ発火する．IOC の返り値は無視してもよい【rte_sws_8021】.

- 全ての受信者への送信を実施する．
- 全ての受信パーティションにおけるデータの書込処理を実施する．

<u>ECU 間連携におけるイベント発火</u>

ECU 間連携の場合，データ送信完了イベントは以下の全ての条件を満たす場合のみ発火する【rte_sws_8023】.

- 全ての受信者への送信を実施する．
- COM からの送信 ACK が完了する．

2.8.4.5　データ受信

データ受信イベント(DataReceivedEvent)が以下の条件を満たす場合，データの受信時にランナブルを起動する〔rte_sws_1135〕【rte_sws_1292】.

- データ受信イベント(DataReceivedEvent)がランナブル(RunnableEntity)と要求側の変数データプロトタイプ(VariableDataPrototype)を参照している．

2.8.4.6　データのタイムアウト

一定時間のデータ受信がないことによるタイムアウトの挙動

データのタイムアウト監視が有効であり，ECU 間連携である場合，RTE は，前回データを受信してからタイムアウト時間が経過したことを検出した際に，受信データセットの値を COM により提供された値に設定し，受信側データセットの状態を「タイムアウト状態」に移行する【rte_sws_5022】．その後，データ受信エラーイベントを起動契機とするランナブルを起動する【rte_sws_a_0069】．

パーティションの停止／再起動にともなうタイムアウトの挙動

データのタイムアウト監視が有効であり，パーティション間連携である場合，送信側パーティションが停止されたらすぐに受信側にタイムアウトを通知し，受信データセットの状態を「タイムアウト状態」に移行する【rte_sws_2710】．その後，データ受信エラーイベントを起動契機とするランナブルを起動する【rte_sws_a_0068】．しかし，A-RTE は，データ受信エラーイベントを起動契機とするランナブルの起動をサポートしない【irte_sws_0028】．

複数のランナブルを起動する場合の実行順番

データのタイムアウト監視において起動する複数の異なるイベントが存在する場合，ランナブルの起動順番は以下とする【nrte_sws_0375】.

<table>
<tr><th>起動順番</th><th>起動対象</th><th>詳細条件</th></tr>
<tr><td>1</td><td>OS タスクにマッピングされているエクスキュータブル
(OS タスク単位で起動する)</td><td>OS タスク内のエクスキュータブルの起動順番は，〔rte_sws_a_0059〕に従う．OS タスク間の起動順番は以下とする．
<table>
<tr><th>起動順番</th><th>詳細条件</th></tr>
<tr><td>1</td><td>OS タスク優先度が高い順</td></tr>
<tr><td>2</td><td>同じ優先度の OS タスクが存在する場合は，OS タスクの名前順(アルファベットの昇順)</td></tr>
<tr><td>3</td><td>同じ優先度の OS タスクが存在し，かつ，イベント設定の場合は，OS イベントの名前順(アルファベットの昇順)</td></tr>
</table>
</td></tr>
</table>

2.8.5　S/R 連携の実現方式

2.8.5.1　1:N 連携の送信

1:N 連携における，1 つの送信側から複数の受信側への送信（以下，1:N 連携送信）の実現方式は，連携のパターンに依存する．

パーティション内連携の 1:N 連携送信

パーティション内連携において，RTEGEN は，RTE の内部で 1:N 連携送信を実現するか，COM を使用して 1:N 連携送信を実現するかを選択してよい【rte_sws_a_0022】.

本 RTE では，RTE 内部で全受信側のバッファにデータを書き込むことで 1:N 連携送信を実現する【nrte_sws_0024】.

パーティション間連携の 1:N 連携送信

IOC は 1:N 連携送信をサポートしない．そのため，パーティション間連携において，RTE が全受信側に対する IOC 用システムサービスを呼び出すことで 1:N 連携送信を実現する【rte_sws_6024】.

ECU 間連携の 1:N 連携送信

ECU 間連携の 1:N 連携送信は，COM を使用して実現する【rte_sws_a_0023】.

ECU 間連携において，複数の受信側への送信を行う 1:N 連携送信には，以下の方式がある.

表 2-24 1:N 連携送信の実現方式

実現方式	内容
PDU による 1:N 連携送信	1 つの I-PDU に対し送信要求を行い, 複数の送信先へのデータ要素の送信を行う.
COM シグナル/COM シグナルグループによる 1:N 連携送信	送信先毎にそれぞれ異なる I-PDU の送信要求を行い，複数の送信先へのデータ要素の送信を行う.

PDU による 1:N 連携送信を使用する場合，1:N 連携送信の実施は通信サービスの責任であるため，RTE は，１つのプリミティブデータ型のデータ要素に対して一度だけ Com_SendSignal を呼び出す. RTE は，通信サービスにて，複数の送信先への送信が行われることを期待する.

COM シグナルによる 1:N 連携送信を使用する場合，RTE は，各送信先に対し，一度ずつ Com_SendSignal を呼び出す.

プリミティブデータ型に対する ECU 間連携の 1:N 連携送信の実現方式の選択

ECU 間連携において，PDU による 1:N 連携送信を用いるか，COM シグナルによる 1:N 連携送信を用いるかは，データ要素に対する COM シグナルのマッピングにより決まるため，RTE の責任は定義内容に応じて Com_SendSignal を呼び出すのみである.

ECU 間連携におけるデータ送信のために，RTE は，データ要素がマッピングされた COM シグナル毎に Com_SendSignal を呼び出す【rte_sws_6023】.

複合データ型に対する ECU 間連携の 1:N 連携送信の実現方式の選択

複合データ型の各要素をアトミックに送受信するために, RTE は COM シグナルグループを使用する【rte_sws_4527】.

ただし，複合データ型が 1 つの COM シグナルにマッピングされた固定長バイト配列の場合，RTE は COM シグナルを使用する【rte_sws_7408】.

COM シグナルグループを使用する場合，RTE は複合データ型の各要素にマッピングされた COM シグナル毎に以下の COM API を呼び出した後，複合データ型にマッピングされた COM シグナルグループに対して ComSendSignalGroup を呼び出す【rte_sws_4526】.

ComShadowSignal API 使用有無 (RteUseComShadowSignalApi)	RTE が呼び出す COM API
true	Com_UpdateShadowSignal
false	Com_SendSignal

2.8.5.2　N:1 連携の送信

N:1 連携における，複数の送信側から 1 つの受信側への送信（以下，N:1 連携送信）の実現方式は，連携のパターンに依存する.

パーティション内連携の N:1 連携送信

パーティション内連携において，RTEGEN は，RTE の内部で N:1 連携送信を実現するか，COM を使用して N:1 連携送信を実現するかを選択してよい【rte_sws_a_0052】．本 RTEGEN では，RTE 内部で受信側のバッファにデータを書き込むことで N:1 連携送信を実現する【nrte_sws_0188】.

パーティション間連携の N:1 連携送信

パーティション間連携において，RTEGEN は，IOC 用システムサービスを呼び出すことで N:1 連携送信を実現する【rte_sws_a_0053】.

ECU 間連携の N:1 連携送信

ECU 間連携において，N:1 連携送信には，以下の方式がある.

実現方式	内容
複数の COM シグナル/COM シグナルグループによる N:1 連携送信	送信側毎に，それぞれ 1 つの COM シグナル/COM シグナルグループを用いてデータを送受信する．
共通の COM シグナル/COM シグナルグループによる N:1 連携送信	複数の送信側から，ある 1 つの COM シグナル/COM シグナルグループを用いて送信されたデータを，受信側で受信する．

複数の COM シグナル/COM シグナルグループによる N:1 連携送信のデータ一貫性保証

COM シグナル/COM シグナルグループ毎に，独立して受信データに対するコールバックが発生する．ある 1 つの受信側に対し複数のコールバックからデータの受信が通知されるため，データの一貫性保証を行う必要がある．

データセマンティクスのデータ要素を受信する受信側 SW-C に対して，複数の送信側 SW-C が ECU 間送信のために異なる COM シグナル/COM シグナルグループを使用する場合は，受信側の RTE はデータ一貫性を確保したうえで，最後に受信したデータを受信側 SW-C に渡さなければならない【rte_sws_3760】．

イベントセマンティクスのデータ要素を使用する受信側 SW-C に対して，複数の送信側 SW-C が ECU 間送信のために異なる COM シグナル/COM シグナルグループを使用する場合は，受信側の RTE はデータ一貫性を確保したうえで，全ての受信データをキューイングしなければならない【rte_sws_3761】．

AUTOSAR 仕様との違い

AUTOSAR 仕様では，複数の送信側が，ECU 間送信のために COM シグナル/COM シグナルグループを共有する場合，送信側の RTE は，これらの COM シグナル/COM シグナルグループにおける COM API が同時に呼び出されないことを保証しなければならないと規定されている【rte_sws_3762】．しかし，同時呼出し防止に要するオーバヘッドを最小化するために，COM API はリエントラントであることを前提とし〔nrte_sws_ext_0005〕，本 RTE では，ある COM シグナル/COM シグナルグループにおいて COM API が同時に呼び出されないことを保証しない【nrte_sws_0167】.

2.8.5.3　データの初期値

S/R 連携を RTE 内部で実現する場合，RTE 内部で実現する受信データセットに初期値を設定する【nrte_sws_0025】．

S/R 連携を IOC の使用によって実現する場合，受信側ポートの所属するパーティションから IOC 用システムサービスを使用して初期値を設定する【nrte_sws_0026】．

S/R 連携を COM の使用によって実現する場合は，初期値は COM により設定されるため，RTE では初期値を設定しない【nrte_sws_0027】．

2.8.5.4　データの無効化

データ無効化は，基本的に COM の提供する COM シグナルの無効化機能を使用して実現する．ただし，COM の機能だけではデータ無効化を実現できない場合には，RTE によりデータ無効化の一部，もしくは全てを実現する．

ECU 内連携におけるデータの無効化

RTEGEN は，ECU 内連携におけるデータ無効化を，RTE 内部で実装するか，COM により実装するかを選択してよい【rte_sws_5025】．本 RTEGEN は，ECU 内連携におけるデータ無効化は，RTE 内部で実現する【nrte_sws_0028】．

ECU 内連携において，無効値受信時処理が keep の場合，RTE は，データの無効化の際に，受信データセットの値に無効値を設定する【rte_sws_5030】．

ECU 内連携において，無効値受信時処理が replace の場合，RTE は，データの無効化の際に，受信データセットの値に初期値を設定する【rte_sws_5049】．

ECU 間連携におけるデータの無効化

ECU 間連携におけるデータの無効化は, 送信側の ECU における無効値の送信, および受信側の ECU における受信の 2 つの処理に分かれる．

無効値の送信は，常に COM で実現する．

無効値の受信は，以下のいずれかの方式で実現する． 無効値の受信をどちらの方式で行うかは，RTE の設定内容に依存して自動的に決定される．

実現方式	内容	実現方式の選択される条件
COM のみによる実現	COM シグナルが持つ無効化機能をそのまま使用する．	ある送信側から値を受信する全受信側ポートの無効値受信時処理が同じ場合，無効値を受信した際の処理は COM の機能を使用して行う【rte_sws_a_0024】.
COM, および RTE による実現	COM シグナルのバッファに受信した無効値を取得し, RTE 内部のバッファに対して無効値の受信処理を行う．	ある送信側から値を受信する，いずれかの受信側ポートの無効値受信時処理の値が異なる場合，無効値を受信した際の処理は RTE 内部で実現する【rte_sws_a_0025】.

以下に，無効値の受信の各実現方式の設定内容例を示す．

・ COM のみによる実現

ある COM シグナルにひもづく全受信側データ要素間で，無効値受信時処理が等しい場合，受信側データ要素と COM シグナルの無効値受信時処理の設定を揃えることができるため，COM のみで実現することが可能である．

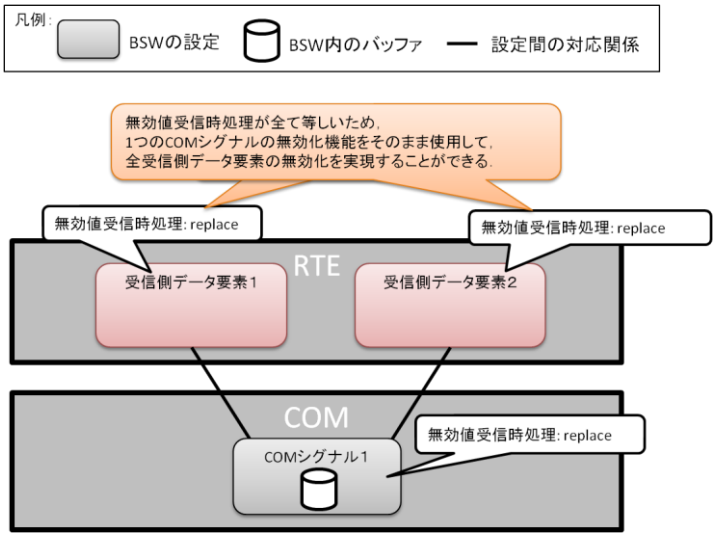

図 2-21 無効値の受信を COM のみで実現する場合の設定

・ COM，および RTE による実現

ある COM シグナルにひもづく受信側データ要素間で，無効値受信時処理が異なる場合，受信側データ要素と COM シグナルの無効値受信時処理の設定を揃えることができないため，COM，および RTE で実現する必要がある．

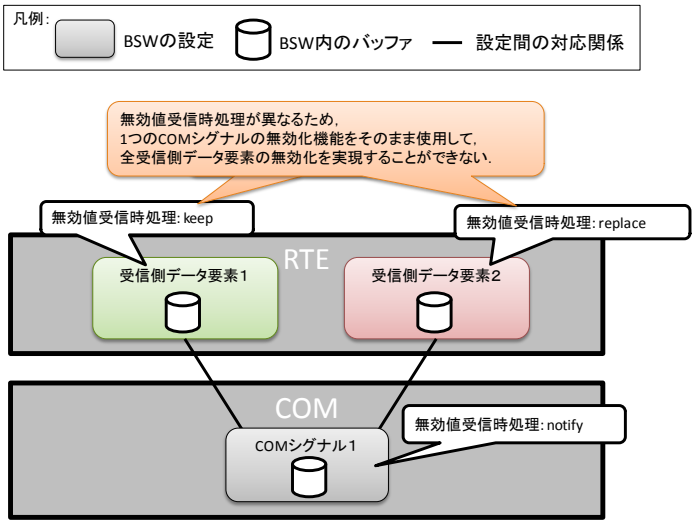

図 2-22 無効値の受信を COM，および RTE で実現する場合の設定

ECU 間連携における無効値の送信

RTE は，データの無効値の送信のために Com_InvalidateSignal を呼び出す【rte_sws_1231】．Com_InvalidateSignal を呼び出す代わりに，無効値を指定して Com_SendSignal を呼び出してもよいものとし，いずれの API を呼び出すかは実装定義とする【nrte_sws_0029】．

ECU 間連携における無効値の受信（COM のみによる実現）

以下の条件を全て満たすデータ要素が無効化された場合，RTE は，受信データセットを COM が受信した値に設定し，受信データセットの状態を「無効状態」に移行する【rte_sws_5026】．

- ECU 間連携である．
- ある送信側から値を受信する受信側ポートの間で無効値受信時処理が全て等しい．
- 受信ポートの無効値受信時処理が keep である．

また〔rte_sws_5026〕の条件を満たす場合，COM は，データ無効化の際に無効値をそのまま保持するように設定されていなければならない【nrte_sws_ext_0013】．

以下の条件を全て満たすデータ要素が無効化された場合，RTE は，受信データセットを COM が受信した値に設定し，受信データセットの状態を「通常状態」に移行する【rte_sws_5048】．

- ECU 間連携である．
- ある送信側から値を受信する受信側ポートの間で無効値受信時処理が全て等しい．
- 受信ポートの無効値受信時処理が replace である．

また〔rte_sws_5048〕の条件を満たす場合，COM は，データ無効化の際に COM シグナルの値を初期値に置換するように設定されていなければならない【rte_sws_ext_a_0003】．

ECU 間連携における無効値の受信（COM，および RTE による実現）

以下の条件を全て満たすデータ要素が無効化された場合，RTE は，受信データセットを無効値に設定し，受信データセットの状態を「無効状態」に移行する【rte_sws_7031】．

- ECU 間連携である．
- ある送信側から値を受信する受信側ポートの間で無効値受信時処理が異なる．
- 受信ポートの無効値受信時処理が keep である．

以下の条件を全て満たすデータ要素が無効化された場合，RTE は，受信データセットを初期値に設定し，受信データセットの状態を「通常状態」に移行する【rte_sws_7032】．

- ECU 間連携である．
- ある送信側から値を受信する受信側ポートの間で無効値受信時処理が異なる．
- 受信ポートの無効値受信時処理が replace である．

RTE が受信した無効値を使用して無効値受信時処理を行うため，COM シグナルのバッファに受信した無効値をそのまま保持しておく必要がある．そのため，〔rte_sws_7031〕，もしくは〔rte_sws_7032〕のいずれかの条件を満たす場合，COM は，データ無効化の際に無効値をそのまま保持するように設定されていなければならない【nrte_sws_ext_0014】．

2.8.5.5　送信 ACK

ECU 間連携において，RTEGEN は，COM により提供される通知機能を使用して，送信 ACK の状態管理を実装する．

2.8.5.6　データのタイムアウト監視

パーティション間連携において，RTEGEN は，RTE 内部で，タイムアウト監視を実装する【rte_sws_a_0026】．

ECU 間連携において，RTEGEN は，COM により提供されるタイムアウト監視機能を使用して，データのタイムアウト監視を実装する【rte_sws_a_0027】．

2.8.5.7　データのフィルタ

ECU 間連携において，RTEGEN は，データのフィルタを COM のフィルタ機能を使用して実現する．パーティション内連携，およびパーティション間連携において，RTE は COM のフィルタ機能を使用できるが，効率性の観点から COM のフィルタ機能を使用せず，RTE でフィルタ機能を実装してもよい【rte_sws_5500】．

本 RTEGEN は，パーティション内連携，およびパーティション間連携におけるデータのフィルタを，RTE 内部で実現する【nrte_sws_0030】．N:1 連携において ECU 間連携とパーティション内連携，あるいはパーティション間連携が混在する場合，フィルタの整合性を取るため，COM のフィルタ機能を使用せず RTE 内部でデータのフィルタを実現する【nrte_sws_0058】．

本 RTE の内部で実現するフィルタでは，パーティションの再起動時に，フィルタで使用するバッファを初期化しない【nrte_sws_0060】．

2.8.5.8　排他方式

RTE 内部で S/R 連携を実現する場合の排他方式は実装定義とする【nrte_sws_0254】．A-RTE の排他方式は以下の通りである【irte_sws_0014】.

<u>実装データ型がプリミティブ/ポインタ実装データ型の場合(コア間連携なし)</u>

データのバッファリング方法	データのタイムアウト監視	データのフィルタ	排他方式
データセマンティクス	なし	なし	(排他しない)
	なし	あり	OS 割込みのブロック
	あり	なし	OS 割込みのブロック
	あり	あり	OS 割込みのブロック
イベントセマンティクス	(関係なし)	(関係なし)	OS 割込みのブロック

実装データ型が配列/構造体/共用体実装データ型の場合(コア間連携なし)

データのバッファリング方法	データのタイムアウト監視	データのフィルタ	排他方式
データセマンティクス	なし	なし	OS 割込みのブロック
	なし	あり	OS 割込みのブロック
	あり	なし	OS 割込みのブロック
	あり	あり	OS 割込みのブロック
イベントセマンティクス	(関係なし)	(関係なし)	OS 割込みのブロック

実装データ型がプリミティブ/ポインタ実装データ型の場合(コア間連携あり)

データのバッファリング方法	データのタイムアウト監視	データのフィルタ	排他方式
データセマンティクス	なし	なし	(排他しない)
	なし	あり	OS スピンロックの獲得
	あり	なし	OS スピンロックの獲得
	あり	あり	OS スピンロックの獲得
イベントセマンティクス	(関係なし)	(関係なし)	OS スピンロックの獲得

実装データ型が配列/構造体/共用体実装データ型の場合(コア間連携あり)

データのバッファリング方法	データのタイムアウト監視	データのフィルタ	排他方式
データセマンティクス	なし	なし	OS スピンロックの獲得
	なし	あり	OS スピンロックの獲得
	あり	なし	OS スピンロックの獲得
	あり	あり	OS スピンロックの獲得
イベントセマンティクス	(関係なし)	(関係なし)	OS スピンロックの獲得

時間パーティショニング機能対応 OS におけるデータ一貫性保証

時間パーティショニング機能対応 OS においては，S/R 連携におけるデータの一貫性を保証しない．データの一貫性を保証することは ECU インテグレータの責任である【nrte_sws_ext_0035】.

非信頼 OSAP に所属する処理単位から排他を実現するためのシステムサービスを使用できないため，本 RTE ではデータの一貫性保証を行わない．

2.8.6　S/R 連携の設定

2.8.6.1　S/R インタフェース，およびデータ要素

S/R インタフェースは，*S/R インタフェース(SenderReceiverInterface)*により指定する．

データ要素は，*S/R インタフェース(SenderReceiverInterface)のデータ要素(dataElement)*により指定する．

2.8.6.2　送信側ポート，および受信側ポート

送信側ポートは，S/R インタフェースを参照する*提供側ポートプロトタイプ(PPortPrototype)*により指定する．

受信側ポートは，S/R インタフェースを参照する*要求側ポートプロトタイプ(RPortPrototype)*により指定する．

2.8.6.3　ポート間の接続関係

送信側ポート-受信側ポート間の接続関係は，送信（提供）側ポート，および受信（要求）側ポートを参照する*アセンブリコネクタ(AssemblySwConnector)*により指定する．RTEGEN は，要求側ポート同士の接続や，提供側ポート同士の接続された設定を拒否【rte_sws_7192】し，コード生成を中止する．

2.8.6.4　データ要素間の接続関係

ECU 内連携におけるデータ要素の接続関係

ECU 内連携において，RTEGEN は，以下のルールに基づき，*S/R インタフェース(SenderReceiverInterface)のデータ要素(dataElement)*の接続関係を判定する【rte_sws_3815】．

- 送信側ポート-受信側ポートの間で，ショートネームが一致するデータ要素を接続されているものとして扱う．
- ポート間で対応するデータ要素が存在しない場合，そのデータ要素は未接続である(接続されているポートがない)ものとして扱う．

ECU 間連携におけるデータ要素の接続関係(プリミティブデータ型)

ECU 間連携において，他 ECU 上のデータ要素との接続関係は，データ要素と *COM シグナル(ComSignal)*のマッピングにより指定する．

プリミティブデータ型のデータ要素と COM シグナルのマッピングは，*S/R 連携シグナルマッピング(SenderReceiverToSignalMapping)*により指定する．データ要素が COM シグナルへマッピングされていない場合，そのデータ要素は未接続である(接続されているポートがない)ものとして扱う【rte_sws_4504】．

データ要素の送受信のため，COM API を呼び出す場合，RTEGEN はデータ要素に対応する *COM シグナル(ComSignal)の COM ハンドル ID(ComHandleId)*を使用する【rte_sws_4505】【rte_sws_3007】．

ECU 間連携におけるデータ要素の接続関係(複合データ型)

複合データ型のデータ要素と COM シグナルグループのマッピングは，*S/R 連携シグナルグループマッピング(SenderReceiverToSignalGroupMapping)*により指定する．複合データ型の各要素(プリミティブ実装データ型)と COM シグナルのマッピングは *S/R 連携複合データ型マッピング*

*(SenderRecCompositeTypeMapping)*により指定する.

データ要素の送受信のため，COM API を呼び出す場合，RTEGEN はデータ要素に対応する *COM シグナルグループ(ComSignalGroup)*の *COM ハンドル ID(ComHandleId)*を使用する【rte_sws_5081】【rte_sws_3008】.

複合データ型の各要素の送受信のため，COM API を呼び出す場合，RTEGEN はデータ要素に対応する *COM グループシグナル(ComGroupSignal)*の *COM ハンドル ID(ComHandleId)*を使用する【rte_sws_5173】.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，データ要素および複合データ型の各要素(プリミティブ実装データ型)が COM シグナルグループ/COM グループシグナルへマッピングされていない場合，そのデータ要素は未接続である(接続されているポートがない)ものとして扱うと規定している【rte_sws_4506】.

しかし，この仕様を厳密に守るとコンフィギュレーションミスを検出できなくなるため，本 RTE ではサポートしない【nrte_sws_0310】.

ECU 間連携における設定制限

1:1 連携，および 1:N 連携の ECU 間連携において，COM と RTE における以下の設定は一致していなければならない【nrte_sws_ext_0017】.

- データの初期値
- データの無効化
- データのタイムアウト監視
- データのフィルタ

2.8.6.5　データの初期値

データの初期値の設定

データの初期値は，*データセマンティクス受信側連携仕様(NonqueuedReceiverComSpec)*，もしくは*データセマンティクス送信側連携仕様(NonqueuedSenderComSpec)*の *初期値(initValue)*により指定する.

送信側と受信側の初期値の共用

あるパーティション内で，送信側と受信側のいずれか一方のみに初期値が指定された場合，RTEGEN は，指定された初期値を送信側，および受信側の両方で使用する【rte_sws_4501】.

送信側と受信側の初期値の優先度

あるパーティション内で，受信側と送信側の両方で初期値が指定された場合，RTEGEN は，受信側で指定された初期値を使用する【rte_sws_4502】.

データの初期値の使用条件

RTEGEN は，以下のいずれかの条件を全て満たす場合，受信側ポートのデータ要素に指定された初期値を使用する【rte_sws_6010】.

- 〔rte_sws_7046〕の初期化条件が満たされている.
- データ要素がデータセマンティクスである.

以上の条件を満たさない場合，初期値は不定となる【rte_sws_a_0028】.

イベントセマンティクスにおける初期値

イベントセマンティクスでは，データを受信していない状態で受信データ取得を行っても無効な値を返すことはないため，初期値は不要となる.

データ要素がイベントセマンティクスの場合，初期値は指定できない【rte_sws_4500】.

2.8.6.6　データのバッファリング

バッファリング方式の設定

データのバッファリング方式は，データ要素の*ソフトウェア実装ポリシ(swImplPolicy)*により指定する．*ソフトウェア実装ポリシ(swImplPolicy)*の設定値とバッファリング方式の対応は以下の通りである【rte_sws_2515】.

設定値	バッファリング方式
queued 以外	データセマンティクス
queued	イベントセマンティクス

受信キューの設定

受信キューのキュー長は，*イベントセマンティクス受信側連携仕様(QueuedReceiverComSpec)*の*キュー長(queueLength)*により指定する.

2.8.6.7　データの無効化

*無効化ポリシ(InvalidationPolicy)*の*無効値受信時処理(handleInvalid)*が keep，もしくは replace に設定されている場合，データ要素のための無効化機能を有効とし．無効値はデータ要素の*無効値(invalidValue)*により指定する【rte_sws_5024】.

無効値処理方式は，受信側の*無効化ポリシ(InvalidationPolicy)*の*無効値受信時処理(handleInvalid)*により指定する【rte_sws_5032】.

2.8.6.8　送信 ACK

*送信側連携仕様(SenderComSpec)*の*送信 ACK 要求(TransmissionAcknowledgementRequest)*が存在する場合，送信側データ要素の送信 ACK は有効となる【rte_sws_1283】.

*送信 ACK 要求(TransmissionAcknowledgementRequest)*の*タイムアウト値(timeout)*が 0 である場合，

タイムアウト監視は実施しない【rte_sws_3758】.

2.8.6.9　データのタイムアウト監視

*データセマンティクスのデータ要素に対し，生存タイムアウト(aliveTimeout)*が存在し，かつ*生存タイムアウト(aliveTimeout)*が 0 より大きい場合，データのタイムアウト監視を有効とする【rte_sws_5020】．*生存タイムアウト(aliveTimeout)*が 0 である場合，タイムアウト監視を無効とする【rte_sws_3759】．

タイムアウト監視の制限

*生存タイムアウト(aliveTimeout)*が存在する場合でも，パーティション内連携である場合，タイムアウト監視は無効とする【rte_sws_5021】．

2.8.6.10 データのフィルタ

RTEGEN は，データセマンティクスのデータ要素に対する*データフィルタ(dataFilter)*が設定されている場合，データのフィルタを有効とする【rte_sws_5503】．

2.8.6.11 データの初期化有無

データ要素の初期化の有無は，以下のコンフィギュレーションにより指定する．

- *変数データプロトタイプ(VariableDataPrototype)の初期値(initValue)*
- *変数データプロトタイプ(VariableDataPrototype)のソフトウェアアドレッシング方式(SwAddrMethod)のセクション初期化ポリシ(sectionInitializationPolicy)*

以下のいずれかの条件を全て満たす場合，データ要素の初期化の際に，データ要素のためのデータ領域を初期化する．いずれの条件も満たさない場合，本 RTE は，データ要素のデータ領域を初期化しない【nrte_sws_0074】．

データ要素を初期化する条件 1【rte_sws_7046】

- *初期値(initValue)*が定義されている．
- *変数データプロトタイプ(VariableDataPrototype)*に対して*ソフトウェアアドレッシング方式(SwAddrMethod)*が定義されていない．

データ要素を初期化する条件 2【rte_sws_3852】

- *初期値(initValue)*が定義されている．
- *変数データプロトタイプ(VariableDataPrototype)*に対して*ソフトウェアアドレッシング方式(SwAddrMethod)*が定義されている．
- *ソフトウェアアドレッシング方式(SwAddrMethod)のセクション初期化ポリシ(sectionInitializationPolicy)*に対する**初期化戦略(RteInitializationStrategy)**が，RTE_INITIALIZATION_STRATEGY_NONE ではない．

2.8.6.12 データの初期化タイミング

データ要素の初期化タイミングは，以下のコンフィギュレーションにより指定する．

- *変数データプロトタイプ(VariableDataPrototype)のソフトウェアアドレッシング方式(SwAddrMethod)のセクション初期化ポリシ(sectionInitializationPolicy)*
- *セクション初期化ポリシ(sectionInitializationPolicy)*に対する**初期化戦略(RteInitializationStrategy)**の設定値に応じたデータの初期化タイミングは以下の通りである【rte_sws_a_0029】．

設定値	データの初期化タイミング
RTE_INITIALIZATION_STRATEGY_AT_DATA_DECLARATION	変数の宣言時.
RTE_INITIALIZATION_STRATEGY_AT_DATA_DECLARATION_AND_PARTITION_RESTART	変数の宣言時，およびパーティションの再起動時.
RTE_INITIALIZATION_STRATEGY_AT_RTE_START_AND_PARTITION_RESTART	RTE の起動時，およびパーティションの再起動時.
RTE_INITIALIZATION_STRATEGY_NONE	なし(初期化を行わない).
RTE_INITIALIZATION_STRATEGY_AT_RTE_START	RTE の起動時【nrte_sws_0056】.

- *変数データプロトタイプ(VariableDataPrototype)のソフトウェアアドレッシング方式(SwAddrMethod)* が定義されていない場合の初期化タイミングは，RTE の起動時である【nrte_sws_0238】.

2.8.6.13 イベントの割り当て

*データ送信完了イベント(DataSendCompletedEvent)*の割り当て

*データ送信完了イベント(DataSendCompletedEvent)*は，*イベントソース(eventSource)*として，提供側データ要素を参照する*変数アクセス(VariableAccess)*を指定する．*データ送信完了イベント(DataSendCompletedEvent)のイベント起動対象ランナブル(startOnEvent)*で指定する．

*データ受信イベント(DataReceivedEvent)*の割り当て

*データ受信イベント(DataReceivedEvent)*は，*データ(data)*として，要求側データ要素を参照する*変数データプロトタイプ(VariableDataPrototype)*のインスタンスを指定する．*データ受信イベント*

*(DataReceivedEvent)のイベント起動対象ランナブル(startOnEvent)*で指定する．

*データ受信エラーイベント(DataReceiveErrorEvent)*の割り当て

*データ受信エラーイベント(DataReceiveErrorEvent)*は，*データ(data)*として，要求側データ要素を参照する*変数データプロトタイプ(VariableDataPrototype)* のインスタンスを指定する．*データ受信エラーイベント(DataReceiveErrorEvent)のイベント起動対象ランナブル(startOnEvent)*で指定する．

2.9　C/S 連携

RTE は，ある SW-C から他 SW-C のサービスの呼出し(C/S 連携)を提供する．C/S 連携では，サービスをオペレーション，サービスの呼出し元をクライアント，サービスの提供元をサーバと呼ぶ．

クライアントは，サービス要求が開始してからサーバからの応答を受け取るまで待ち合わせを行う，同期連携をサポートする．サービスの結果は，C/S 連携の API の返り値として取得できる．

2.9.1　C/S 連携の構成

C/S 連携は以下の要素から構成される．

C/S インタフェース

C/S 連携におけるポートインタフェースであり，SW-C 間のオペレーション呼出し内容を規定するインタフェース仕様を表す．

アプリケーションエラー

C/S インタフェースによるオペレーション呼出しで発生しうるエラーを表す．

オペレーション

C/S 連携のインタフェース要素であり，C/S 連携で実際に呼び出すサービスを表す．

オペレーション引数

オペレーション呼出しの際に渡す引数を表す．クライアントからのオペレーションの入力情報の引渡し，およびオペレーションの処理結果のクライアントへの引渡しに使用される．

オペレーション使用アプリケーションエラー

オペレーションにおいて発生しうるアプリケーションエラーを表す．

ポート定義引数値

オペレーション呼出しの際に渡す固定の引数値を表す．サーバへサーバポート固有の入力情報を引渡すために使用される．

サーバポート

C/S 連携における提供側ポートであり，オペレーションを提供する SW-C のポート．

クライアントポート

C/S 連携における要求側ポートであり，オペレーションを呼び出す SW-C のポート．

サーバオペレーション

サーバポートにおけるオペレーションのインスタンス.

クライアントオペレーション

クライアントポートにおけるオペレーションのインスタンス．オペレーション呼出しのための API はクライアントオペレーション毎に生成される.

サーバ SW-C

C/S 連携における提供側 SW-C であり，C/S インタフェースを介してオペレーションを提供する SW-C.

クライアント SW-C

C/S 連携における要求側 SW-C であり，C/S インタフェースを介してオペレーションを呼び出す SW-C.

サーバランナブル

オペレーション呼出しイベントにより起動し，オペレーションの処理を行うランナブル．オペレーションの処理内容は，本ランナブルの処理内容として定義される.

2.9.2　C/S 連携の種別

2.9.2.1　C/S 連携の多重度

RTE は，以下の多重度の C/S 連携をサポートする．連携の多重度については，2.4.2.2 節を参照.

- 1:0 連携，0:1 連携〔rte_sws_1329〕
- 1:1 連携
- N:1 連携【rte_sws_4519】

以上のいずれのケースについても，RTE は，クライアントポートに対し 1 つのオペレーション要求 API を提供する.

複数のクライアントが同一のオペレーションを使用する場合，RTE は，オペレーションの結果が適切なクライアントに送られることを保証する【rte_sws_4516】.

2.9.2.2　オペレーション数

C/S インタフェースは，1 つ以上のオペレーションを含む.

RTE は，オペレーション毎異なる RTE API を提供し，オペレーション毎に独立してサービスの呼出しを行う【rte_sws_5110】【rte_sws_4517】.

2.9.2.3　ポート定義引数値

C/S インタフェースに含まれない，サーバ固有の情報（BSW の API が要求する固定の ID 等）がサ

ービスの処理に必要となる場合がある．RTE は，これらの情報をポート定義引数値としてサーバに提供する．

ポート定義引数は，サーバポート毎に，コンフィギュレーションにより設定できる．

2.9.3　C/S 連携の操作

2.9.3.1　オペレーション呼出し

RTE は，オペレーション毎にオペレーション呼出しのための API(Rte_Call)を提供する．クライアント SW-C は，提供された API を使用することで，オペレーション呼出しを行う．オペレーション呼び出しの引数と返り値は，アトミックに扱う【rte_sws_4518】.

オペレーション呼出しの挙動

Rte_Call によりオペレーション呼出しが開始された場合の挙動は以下の通りである【nrte_sws_0076】.

(1) RTE は，API により指定されたオペレーション引数の値をサーバポートに伝搬する．
(2) サーバポートに値が伝搬されると，RTE は，サーバランナブルに伝搬されたオペレーション引数の値，およびポート定義引数値を渡して起動する．
(3) サーバランナブルは，渡された引数に従ってオペレーションの処理を行い，処理結果の値をサーバランナブルのオペレーション引数，および返り値として返す．
(4) RTE は，サーバランナブルのオペレーション引数の値，および返り値をクライアントポートに伝搬する．
(5) クライアントポートに値が伝搬されると，RTE は，オペレーション呼出し元の Rte_Call の引数に伝搬されたオペレーション引数の値を設定する．そして，伝搬された返り値を API の返り値として返す．

サーバランナブルへの引数渡しの順序

オペレーション呼出しによりサーバランナブルを起動する際，RTE は，以下の順序で引数値を指定して，サーバランナブルを呼び出す【rte_sws_1360】.

(1) 全てのポート定義引数値(ポート定義引数値の定義順)
(2) API により指定された全てのオペレーション引数の値(オペレーション引数の定義順)

オペレーション呼出しの制限

サーバランナブルはオペレーション呼出し API からのみ起動する【rte_sws_6019】．サーバランナブルはオペレーション呼出し API からのみ起動でき，RTE 以外のモジュールから関数呼び出しにより起動してはならない【rte_sws_a_0030】．本 RTE のオペレーション呼び出し範囲は以下の通りである【nrte_sws_0050】.

SW-C 間の連携パターン	SW-C 所属 パーティションの権限		サポート有無
	クライアント	サーバ	
パーティション内連携	非信頼	(同左)	○
	信頼	(同左)	○
パーティション間連携 (コア内)	非信頼	非信頼	×
		信頼	○
	信頼	非信頼	×
		信頼	○
コア間連携	－	－	×
ECU 間連携	－	－	×

オペレーションの並行呼出し

各クライアントは，サーバの, あるオペレーションを並行して呼び出すことができる．RTE は，サーバのオペレーションの並行起動要求をサポートする【rte_sws_4520】.

オペレーション呼出しのデータ一貫性保証

RTE によるオーバヘッドを減らすため，オペレーション呼出しがサーバランナブルの直接関数起動として実現される場合，RTE は，参照渡しされるパラメータのコピーを作成せず, 参照パラメータのデータ一貫性を保証することはアプリケーションの責任である【rte_sws_7008】.

2.9.4　C/S 連携の実現方式

2.9.4.1　オペレーション呼出し

本 RTE における，オペレーション呼出し API(Rte_Call)からのサーバランナブル起動は以下の方法で実現する【nrte_sws_0031】.

SW-C 間の連携パターン	SW-C 所属 パーティションの権限		実現方法
	クライアント	サーバ	
パーティション内連携	非信頼	(同左)	直接関数起動
	信頼	(同左)	直接関数起動
パーティション間連携 (コア内)	非信頼	非信頼	－
		信頼	OS 信頼関数経由での直接関数起動
	信頼	非信頼	－
		信頼	直接関数起動
コア間連携	－	－	－
ECU 間連携	－	－	－

2.9.5　C/S 連携の設定

2.9.5.1　C/S インタフェース，オペレーション，およびアプリケーションエラー

C/S インタフェースは，*C/S インタフェース(ClientServerInterface)*により指定する．

オペレーションは，*C/S インタフェース(ClientServerInterface)*の*オペレーション(operation)*により指定する．

アプリケーションエラーは，*C/S インタフェース(ClientServerInterface)*の*発生可能エラー(possibleError)*により指定する．

2.9.5.2　クライアント，およびサーバポート

クライアントポートは，C/S インタフェースを参照する*要求側ポートプロトタイプ(RPortPrototype)*により指定する．

サーバポートは，C/S インタフェースを参照する*提供側ポートプロトタイプ(PPortPrototype)*により指定する．

2.9.5.3　オペレーション引数

オペレーション引数は，オペレーションの*オペレーション引数(argument)*により指定する．

2.9.5.4　オペレーション使用アプリケーションエラー

オペレーション使用アプリケーションエラーは，オペレーションの*発生可能エラー(possibleError)*により指定する．

2.9.5.5　ポート定義引数値

ポート定義引数値は，サーバポートを参照する*ポート API オプション(PortAPIOption)*の*ポート定義引数値(portArgValue)*により指定する．

2.9.5.6　サーバランナブル

サーバランナブルは，オペレーションを参照する*オペレーション呼出しイベント(OperationInvokedEvent)*で起動する*ランナブル(RunnableEntity)*により指定する【rte_sws_5163】.

2.9.5.7　ポート間の接続関係

クライアントポート-サーバポート間の接続関係は，クライアント(要求側)ポート，およびサーバ(提供側）ポートを参照する*アセンブリコネクタ(AssemblySwConnector)*により指定する．RTEGEN は，要求側ポート同士の接続や，提供側ポート同士の接続された設定を拒否〔rte_sws_7192〕し，コード生成を中止する.

2.9.5.8　オペレーション間の接続関係

RTEGEN は，以下のルールに基づき，*C/S インタフェース(ClientServerInterface)*の*オペレーション(operation)*の接続関係を判定する【rte_sws_3818】.

- クライアントポート-サーバポートの間で，ショートネームが一致するオペレーションを接続されているものとして扱う.
- ポート間で対応するオペレーションが存在しない場合，そのオペレーションは未接続である(接続されているポートがない)ものとして扱う.

2.10 ランナブル間連携

RTE は，SW-C 内のランナブル間の送受信機能として，IRV(InterRunnableVariables)を提供する【rte_sws_3589】．IRV は， SW-C 内のランナブルから送信され，同一 SW-C 内のランナブルで受信される．ランナブル間連携は双方向の連携であるため， 1 つの IRV で送信，および受信ができる．

2.10.1 ランナブル間連携の構成

ランナブル間連携は以下の要素から構成される．

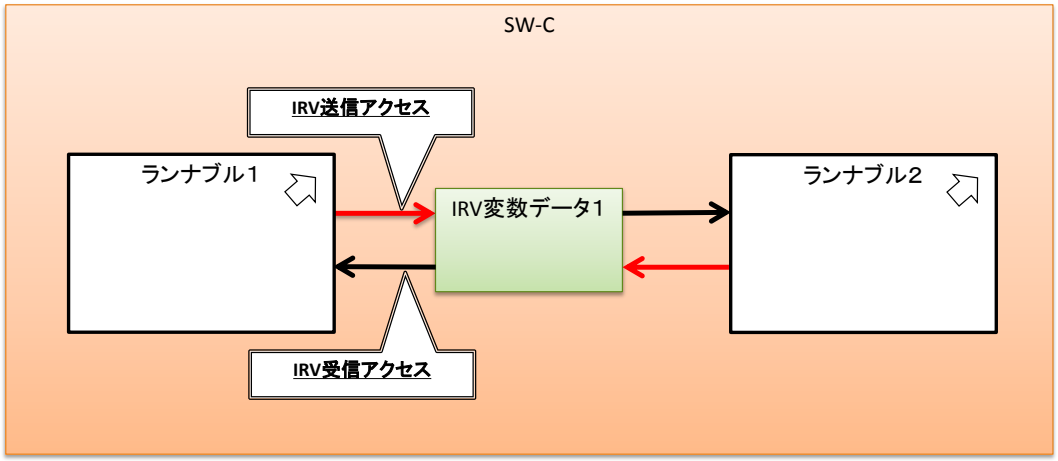

図 2-23 ランナブル間連携の構成

IRV 送信アクセス

ランナブル間連携において，ランナブルがデータ送信するためのアクセス設定．データ送信のための API はこの単位で生成される．

IRV 受信アクセス

ランナブル間連携において，ランナブルがデータ受信するためのアクセス設定．データ受信のための API はこの単位で生成される．

IRV 変数データ

ランナブル間連携における，IRV の変数データのインスタンス．

IRV 初期値

ランナブルが IRV 変数データを受信する前に，データを取得しようとした際に返される値．

2.10.2 ランナブル間連携の操作

2.10.2.1 データ送信

RTE は，IRV 変数データの IRV 送信アクセス毎に，データ送信のための API(Rte_IrvWrite)を提供する．

<u>データ送信の挙動</u>

Rte_IrvWrite によりデータ送信が開始された場合の挙動は以下の通りである【nrte_sws_0261】．

(1) 送信対象データを IRV 変数データに設定する．

2.10.2.2 受信データ取得

RTE は，IRV 変数データの IRV 受信アクセス毎に，受信データ取得のための API(Rte_IrvRead)を提供する．

<u>受信データ取得の挙動</u>

Rte_IrvRead により受信データ取得が開始された場合の挙動は以下の通りである【nrte_sws_0262】.

(1) IRV 変数データを API の返り値または OUT 引数に設定する．

2.10.3 ランナブル間連携のデータ一貫性

RTE は，同じ IRV を使用するランナブル間の連携について，データ一貫性を保証する【rte_sws_3516】【rte_sws_3519】．

2.10.4 ランナブル間連携の実現方式

2.10.4.1 データ送信

本 RTE では，IRV 変数データ毎の RTE 内部バッファにデータを書き込むことでデータ送信を実現する【nrte_sws_0263】．

2.10.4.2 排他方式

RTE 内部でランナブル間連携を実現する場合の排他方式は実装定義とする【nrte_sws_0264】. A-RTE の排他方式は以下の通りである【irte_sws_0020】.

実装データ型	排他方式
プリミティブ実装データ型	(排他しない)
配列実装データ型	OS 割込みのブロック
構造体実装データ型	OS 割込みのブロック
共用体実装データ型	OS 割込みのブロック
ポインタ実装データ型	(排他しない)
再定義実装データ型	再定義実装データ型が参照する実装データ型の型に従う

時間パーティショニング機能対応 OS におけるデータ一貫性保証

時間パーティショニング機能対応 OS においては，ランナブル間連携におけるデータの一貫性を保証しない．データの一貫性を保証することは ECU インテグレータの責任である【nrte_sws_ext_0036】.

非信頼 OSAP に所属する処理単位から排他を実現するためのシステムサービスを使用できないため，本 RTE ではデータの一貫性保証を行わない.

2.10.5 ランナブル間連携の設定

2.10.5.1 IRV 送信アクセス

IRV 送信アクセスは，*ランナブル(RunnableEntity) の書き込みローカル変数(writtenLocalVariable)の変数アクセス(VariableAccess)*により指定する.

2.10.5.2 IRV 受信アクセス

IRV 受信アクセスは，*ランナブル(RunnableEntity) の読み込みローカル変数(readLocalVariable)の変数アクセス(VariableAccess)*により指定する.

2.10.5.3 IRV 変数データ

IRV 変数データは，*SWC 内部振る舞い(SwcInternalBehavior)の直接ランナブル変数(explicitRunnableVariable)の変数データプロトタイプ(VariableDataPrototype)*により指定する.

2.10.5.4 IRV 変数データとの接続

IRV 変数データとの接続は，IRV 送信/受信アクセスの *AUTOSAR 変数参照(AutosarVariableRef)*により指定する.

2.10.5.5 IRV 変数データの初期値

IRV 初期値は，IRV 変数データの *初期値(initValue)*により指定する.

2.10.5.6 IRV 変数データの初期化タイミング

〔rte_sws_7046〕で規定される初期化条件を満たす場合，RTE は，IRV 変数データの値を IRV 初期

値で初期化する【rte_sws_7187】.

2.11 モード連携

SCHM は，エクスキュータブルのモードを管理する機能として，以下のモード連携機能を提供する．

- モードの遷移に伴いエクスキュータブルを起動する
- ある特定のモードにおいてエクスキュータブルの起動を有効化，もしくは無効化する

2.11.1 モード連携の構成

モード連携は以下の要素から構成される．

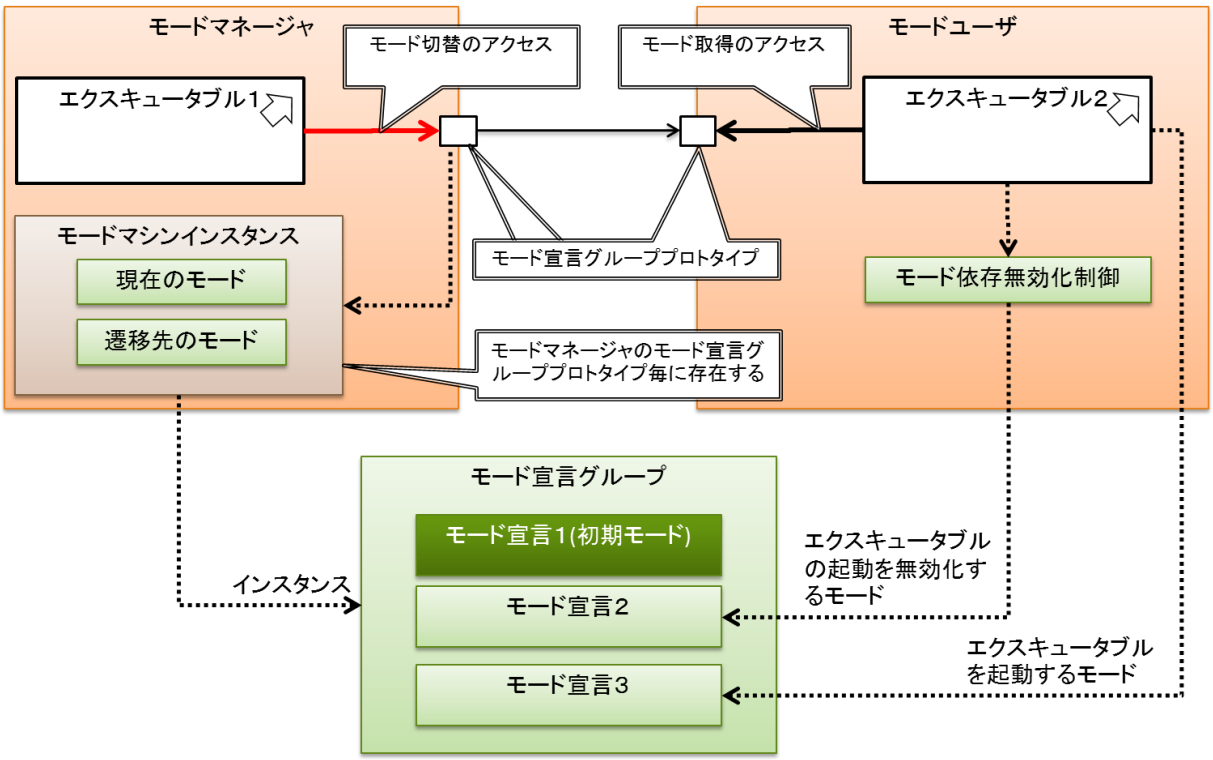

図 2-24 モード連携の構成

モードマネージャ

モードを切替する BSWM を表す．

モードユーザ

モード切替により以下の機能を利用する BSWM を表す．

- モード切替契機でのエクスキュータブルの起動
- モード依存無効化制御によるエクスキュータブルの起動抑制

モード宣言

モード宣言グループの個々のモードを表す．

モード宣言グループ

モード宣言の集合を表す.

モード宣言グループプロトタイプ

モード宣言グループの実体を表す.

モードマシンインスタンス

モード宣言グループプロトタイプの状態マシンを表す．モードマネージャのモード宣言グループプロトタイプ毎に存在する.

要求モード

モード切替通知により，モードマネージャが要求するモードを表す.

初期モード

モードマシンインスタンスのモード(現在のモード，および遷移先のモード)の初期値を表す.

現在のモード

モードマシンインスタンスの現在のモードを表す.

遷移先のモード

モードマシンインスタンスの遷移先のモードを表す．モードマシンインスタンスがモード遷移中の場合のみ，現在のモードと異なる.

モード依存無効化制御

現在のモードまたは遷移先のモードが，ある特定のモードの場合，エクスキュータブルの起動を無効化する制御を表す.

モード取得のアクセス

モード連携において，エクスキュータブルがモード取得するためのアクセス設定．モード取得のための API はこの単位で生成される.

モード切替のアクセス

モード連携において，エクスキュータブルがモード切替するためのアクセス設定．モード切替のための API はこの単位で生成される.

2.11.2 モード連携の種別

2.11.2.1 モード連携の多重度

SCHM は，以下の多重度のモード連携をサポートする．

- 1:0 連携，0:1 連携
- 1:1 連携
- 1:N 連携【rte_sws_2566】【rte_sws_7539】

以上のいずれのケースについても，SCHM はモードマネージャ，およびモードユーザに対して，モード連携用の API を提供する．

モードマネージャはモードユーザの数に関わらず, モード切替するために提供された 1 つの API を呼び出すのみでよい．

2.11.2.2 モード連携のバッファリング

SCHM は，モードマシンインスタンス単位でモード切替通知を行い，モード切替要求のバッファリング方式はイベントセマンティクスである【rte_sws_2624】【rte_sws_2718】.

イベントセマンティクスによるバッファリング

RTEGEN は，モードマネージャのモードマシンインスタンス毎に，モード切替要求キューを実装し，キューは FIFO で実現する【rte_sws_2719】．キューが一杯の状態で，新しいモード切替通知が行われた場合は，SCHM はその要求を無視する【rte_sws_2720】．この場合，SchM_Switch はエラーを返す〔rte_sws_7259〕．モード切替要求をキューから削除するタイミングは，モード切替が終了したときである【rte_sws_2721】.

2.11.3 モード連携の状態

2.11.3.1 モード切替要求キューの状態

モードマシンインスタンスのモード切替要求キューの状態遷移を以下に定義する【nrte_sws_0316】.

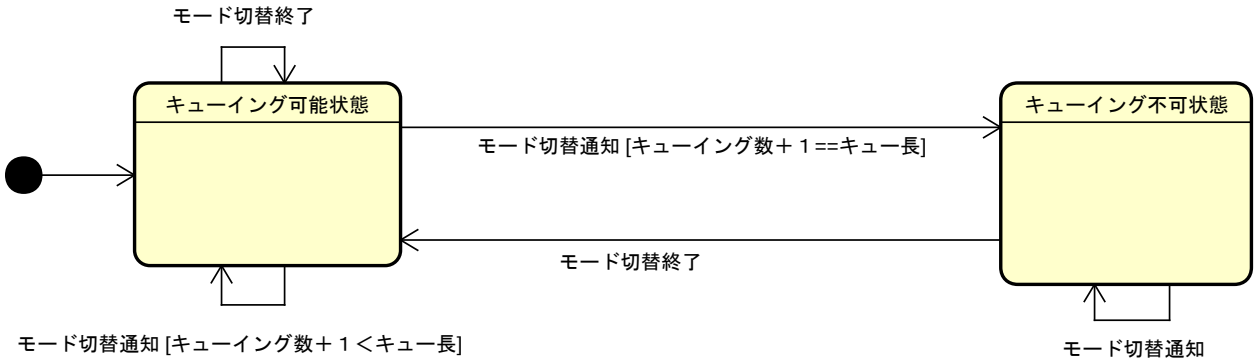

図 2-25 モード切替要求キューの状態遷移図

表 2-25 モード切替要求キューの状態

キュー状態	説明
キューイング可能状態	新しい要求を追加できる状態.
キューイング不可状態	キューが一杯のため，新しい要求を追加できない状態.

表 2-26 モード切替要求キューの遷移

遷移イベント	説明
モード切替通知	モード切替通知が発生した.
モード切替終了	モード切替が終了した.

表 2-27 モード切替要求キューの状態遷移表

遷移イベント キューの状態	モード切替通知	モード切替終了
キューイング可能状態	[キューイング数＋１==キュー長] キューイング不可状態 [キューイング数＋１＜キュー長] キューイング可能状態	キューイング可能状態
キューイング不可状態	キューイング不可状態	キューイング可能状態

2.11.4 モード連携の操作

モード切替の実現にあたり，SCHM は以下を前提とする.

- エクスキュータブルが起動中は，各モード宣言グループプロトタイプのモードは，１つのモードまたは１つのモード遷移のみがアクティブであることを前提とする【rte_sws_ext_2542】. これは，モードマシンインスタンスがネストされたモードを持たないことを意味する.
- モードマシンインスタンスの初期化終了後，モード宣言グループプロトタイプの初期モードへの遷移が実行される〔rte_sws_2544〕. このモード遷移では，モード依存無効化制御およびモード入場時起動エクスキュータブルの起動も実行される.

モード連携では，モード切替の手順として，以下の手順がある.

- 同期モード切替手順
- 非同期モード切替手順

モード切替手順は，モードユーザのコンフィギュレーションにより決まる.

モードユーザの非同期モードサポートのコンフィギュレーション状態	選択されるモード切替手順
全モードがサポートしない	同期モード切替手順【rte_sws_7150】
一部モードユーザがサポートする	
全モードユーザがサポートする	非同期モード切替手順【rte_sws_7151】

AUTOSAR 仕様との違い

AUTOSAR 仕様では，同期モード切替手順〔rte_sws_7150〕を規定しているが，本 SCHM では同期モード切替手順はサポートしない【nrte_sws_0318】.

2.11.4.1 モード切替通知

SCHM は，モードマネージャ毎に，モード切替通知のための API(SchM_Switch)を提供する．モードマネージャは，提供された API を使用することができる．要求モードと現在のモードが同じ場合でもモード切替通知は実行される【rte_sws_2669】. モード切替は非同期で通知される【rte_sws_2508】.

モード切替通知の挙動(非同期モード切替手順)

SchM_Switch によりモード切替が開始された場合の挙動は以下の通りである.

(1) モードマシンインスタンスのモード切替要求キューの状態，およびキューイング数に応じて以下の処理を行う【rte_sws_2667】.

キューの状態	キューイング数	処理
キューイング可能状態	0	要求モードをキューイングし，〔nrte_sws_0316〕に従って状態遷移する(遷移イベント：モード切替通知).
	1 以上	要求モードをキューイングし，〔nrte_sws_0316〕に従って状態遷移する(遷移イベント：モード切替通知). 以降の処理は実施せず，処理終了する.
キューイング不可状態	(関係なし)	以降の処理は実施せず，エラーを返す〔rte_sws_7259〕.

(2) 以下の処理を実行する【rte_sws_2667】【rte_sws_2665】.

処理順	処理内容	現在のモード	遷移先のモード	起動するエクスキュータブル
1.	遷移先のモードを変更【rte_sws_2661】	(変更なし)	要求モード	(なし)
2.	エクスキュータブル起動【rte_sws_2562】	(変更なし)	(変更なし)	モード退場時起動エクスキュータブル
3.	エクスキュータブル起動【rte_sws_2707】	(変更なし)	(変更なし)	モード遷移時起動エクスキュータブル
4.	エクスキュータブル起動【rte_sws_2564】	(変更なし)	(変更なし)	モード入場時起動エクスキュータブル
5.	現在のモードを変更【rte_sws_2563】	要求モード	(変更なし)	(なし)

(3) 要求モードをキューから削除し，〔nrte_sws_0316〕に従って状態遷移（遷移イベント：モード切替終了)後，キューイング数に応じて以下の処理を行う【rte_sws_2668】.

キューイング数	処理
0	処理終了する.
1 以上	キューの先頭要素を要求モードとして，(2)から処理を再実施する.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，〔rte_sws_2665〕において起動するエクスキュータブルとして，モード退場時起動エクスキュータブル〔rte_sws_2562〕およびモード遷移時起動エクスキュータブル〔rte_sws_2707〕を規定しているが，本 SCHM ではサポートしない【nrte_sws_0317】.

2.11.4.2 エクスキュータブル起動

SCHM は，2.5.4 節の実現方式に従ってエクスキュータブルを起動する．ただし，モード依存無効化制御を設定しているエクスキュータブルは，以下に示す条件の場合，エクスキュータブルの起動を抑止する【rte_sws_7530】.

モード依存無効化制御対象モードとの一致有無		エクスキュータブル起動
現在のモード	遷移先のモード	
一致しない	一致しない	起動する
一致しない	一致する	起動しない
一致する	一致しない	起動しない
一致する	一致する	起動しない

エクスキュータブルの起動抑止では，SCHM は実行中のエクスキュータブルの停止は行わない【rte_sws_7531】.

RTEGEN は，以下の条件を満たす場合，モード切替通知でのエクスキュータブルの直接関数起動をサポートする【rte_sws_7173】.

- モード切替の手順が非同期モード切替手順である〔rte_sws_7151〕
- モードマネージャとモードユーザが同じパーティションに配置されている
- 直接関数呼び出し条件を満たしている〔rte_sws_a_0064〕

複数のエクスキュータブルを起動する場合の実行順番

*モード宣言(ModeDeclaration)*を参照する複数の異なる*モード切替イベント(BswModeSwitchEvent)*が存在する場合，エクスキュータブルの起動順番は以下とする【nrte_sws_0333】.

<table>
<tr><th>起動順番</th><th>起動対象</th><th>詳細条件</th></tr>
<tr><td>1</td><td>直接関数起動するエクスキュータブル</td><td>モード切替イベント(BswModeSwitchEvent)のエクスキュータブルの名前順(アルファベットの昇順).</td></tr>
<tr><td>2</td><td>OS タスクにマッピングされているエクスキュータブル
(OS タスク単位で起動する)</td><td>OS タスク内のエクスキュータブルの起動順番は，〔rte_sws_a_0059〕に従う．OS タスク間の起動順番は以下とする．
<table>
<tr><th>起動順番</th><th>詳細条件</th></tr>
<tr><td>1</td><td>OS タスク優先度が高い順</td></tr>
<tr><td>2</td><td>同じ優先度の OS タスクが存在する場合は，OS タスクの名前順(アルファベットの昇順)</td></tr>
</table></td></tr>
</table>

2.11.4.3 モード取得

SCHM は，モードマシンインスタンスのモード取得のための API(SchM_Mode)を提供する．モードマネージャ，およびモードユーザは，提供された API を使用することができる．

モード取得の挙動

SchM_Mode によりモード取得が開始された場合，SCHM はモードマシンインスタンスのモードとして以下の値を API の返り値に設定する〔rte_sws_7262〕．

現在のモードと遷移先モードとの一致有無	API の返り値
一致しない	モード遷移中のステータス〔rte_sws_7293〕
一致する	現在のモード〔rte_sws_7294〕

2.11.4.4 モードマシンインスタンスの初期化

SCHM は，SchM_Init において，全てのモードマシンインスタンスを初期モードへ遷移させる．【rte_sws_7532】．マルチコア構成の ECU の場合，初期化対象は，SchM_Init の呼び出し元のコアにマッピングされているモードマシンインスタンスである【nrte_sws_0336】.

モードマシンインスタンスの初期化の挙動

すべてのモードインスタンスに対して，以下の処理を実行する．

処理順	処理内容	現在のモード	遷移先のモード	起動するエクスキュータブル
1.	遷移先のモードを変更【rte_sws_2661】	初期モード	初期モード	(なし)
2.	エクスキュータブル起動【rte_sws_2564】	(変更なし)	(変更なし)	モード入場時起動エクスキュータブル

2.11.5 モード連携の実現方式

2.11.5.1 モード切替通知

本 RTE では，モード切替通知で使用する現在のモード，および遷移先のモードはモードマシンインスタンス毎の RTE 内部バッファで実現する【rte_sws_7540】．

モード切替通知の連携範囲

SCHM のモード切替通知の連携範囲は以下の通り【nrte_sws_0320】.

BSWM 間の連携パターン	BSWM 所属パーティションの権限		サポート有無
	モードマネージャ	モードユーザ	
パーティション内連携	非信頼	(同左)	×
	信頼	(同左)	○
パーティション間連携 (コア内)	非信頼	非信頼	×
		信頼	×
	信頼	非信頼	×
		信頼	×
コア間連携	－	－	×
ECU 間連携	－	－	× 【rte_sws_a_0065】

2.11.5.2 排他方式

モードマシンインスタンスを実現する場合の排他方式は実装定義とする【nrte_sws_0319】. A-RTE の排他方式は OS 割込みのブロックである【irte_sws_0025】.

2.11.6 モード連携の設定

2.11.6.1 モード宣言

モード宣言は，*モード宣言グループ(ModeDeclarationGroup)* の *モード宣言(modeDeclaration)*により指定する.

2.11.6.2 モード宣言グループ

モード宣言グループは，*モード要求型マップ (ModeRequestTypeMap)* の *モードグループ (modeGroup)*により指定する.

2.11.6.3 モード宣言グループの型

モード宣言グループの型は，*モード要求型マップ (ModeRequestTypeMap)* の *実装データ型 (implementationDataType)*により指定する.

2.11.6.4 モード要求型マップ

モード要求型マップは，*内部振る舞い(InternalBehavior)*の *データ型マッピング(dataTypeMapping)* の *モード要求型マップ(modeRequestTypeMap)*により指定する.

2.11.6.5 モード宣言グループプロトタイプ(SCHM)

SCHM のモード宣言グループプロトタイプは，*BSWM ディスクリプション(BswModuleDescription)*

*の提供モードグループ(providedModeGroup)*または*要求モードグループ(requiredModeGroup)*により指定する．RTEGEN は，*提供モードグループ(providedModeGroup)*により指定されたモード宣言プロトタイプ毎に，モードマシンインスタンスを割り当てる【rte_sws_7534】.

モードマネージャは,*提供モードグループ(providedModeGroup)*の BSWM である.モードユーザは,*要求モードグループ(requiredModeGroup)*の BSWM である．モードマネージャが使用するモード宣言は，常に遷移先のモードを表す【rte_sws_ 7541】.

2.11.6.6 モード切替の接続関係(SCHM)

RTEGEN は，*提供モードグループ(providedModeGroup)*と*要求モードグループ(requiredModeGroup)*が **BSW 要求モードグループ接続(RteBswRequiredModeGroupConnection)**で接続されている場合，モード連携の接続があると判断する【rte_sws_7538】.

モードマシンインスタンスのモード切替のアクセス設定は，*BSWM エンティティ(BswModuleEntity)のモードグループ管理(managedModeGroup)*により指定する.モードマシンインスタンスのモード取得のアクセス設定は，*BSWM エンティティ(BswModuleEntity)のモードグループ参照(accessedModeGroup)*により指定する.

2.11.6.7 モード切替通知によるエクスキュータブル起動設定(SCHM)

SCHM のモード切替通知で起動する BSW スケジューラブルは，*モード切替イベント(BswModeSwitchEvent)のイベント起動対象 BSW スケジューラブル(startsOnEvent)*で指定する．起動要因モードは，*モード切替イベント(BswModeSwitchEvent)の起動(activation)*, および*モード(mode)*で指定する.

2.11.6.8 モード依存無効化制御(SCHM)

モード依存無効化制御で指定するモードは，*BSW イベント(BswEvent)の無効化モード(disabledInMode)*により指定する.

2.11.6.9 初期モード

初期モードは，*モード宣言グループ(ModeDeclarationGroup) の初期モード(initialMode)*で指定する.

2.11.6.10 モード切替要求キュー長(SCHM)

モード切替要求キュー長は，*BSW モード送信ポリシー(BswModeSenderPolicy)のキュー長(queueLength)*により指定する.

2.11.6.11 モード切替の手順(SCHM)

モード切替の手順は，*BSW モード受信ポリシー(BswModeReceiverPolicy)の非同期モード切替サポート(supportsAsynchronousModeSwitch)*により指定する.

2.12 排他エリア

RTE/SCHM は，SW-C/BSWM による排他制御を行うための仕組みとして，排他エリアを提供する．排他エリアは，データ一貫性を保証するためのクリティカルセクションを実現し，同時アクセスをブロックする目的で使用する．排他エリアは，SW-C/BSWM のデータ一貫性保証のために使用することができる．

排他エリアを用いたデータ一貫性の保証のためには，同一のデータにアクセスする各エクスキュータブルが同じ排他エリアを獲得するようにコンフィギュレーションを行った上で，エクスキュータブルから排他エリアへ入退場する必要がある．

あるエクスキュータブルが排他エリアに入場している間，その他のエクスキュータブルによる排他エリアへの入場が起こらないように排他される．排他エリアに入場中のエクスキュータブルが排他エリアから退場することで，排他が解除される．

なお，本節では，エクスキュータブルが使用する排他エリアの仕様が，ランナブルと BSWM エンティティとで共通する場合は，エクスキュータブルとして説明し，仕様差分がある場合のみ個別に説明する．

2.12.1 排他エリアの構成

排他エリアは，以下の要素から構成される．

排他エリア

エクスキュータブル実行のクリティカルセクション．排他エリアにあるエクスキュータブルが入場している間は，その他のエクスキュータブルの排他エリアへの入場は行われないよう排他される【rte_sws_3500】【rte_sws_7522】．

使用 OS リソース

排他エリア実現メカニズム「OS リソースの獲得」を適用する場合に，データ一貫性保証のために使用する OS リソース．

使用 OS スピンロック

排他エリア実現メカニズム「OS スピンロックの獲得」を適用する場合に，データ一貫性保証のために使用する OS スピンロック．

2.12.2 排他エリアの種別

2.12.2.1 排他エリア入退場方法

排他エリアへの入退場方法として，以下の 2 つの方法を提供する．2 つの方法は排他的ではなく，組み合わせて使用できる．各入退場方法を有効とするかは，コンフィギュレーションにより選択できる．

入退場方法	内容
明示的入退場	・ランナブルの場合 SW-C 開発者が，排他エリアへの入退場 API(Rte_Enter，および Rte_Exit)を明示的に呼び出す方法．Rte_Enter で排他エリアへの入場を行い，Rte_Exit で排他エリアからの退場を行う． ・BSWM エンティティの場合 BSWM 開発者が，排他エリアへの入退場 API(SchM_Enter，および SchM_Exit)を明示的に呼び出す方法. SchM_Enter で排他エリアへの入場を行い，SchM_Exit で排他エリアからの退場を行う．
エクスキュータブル実行時入退場	・ランナブルの場合 SW-C 開発者が，システム設計時にランナブル実行中は排他エリア内での動作となることを定義し，RTE に保証させる方法．RTE は，ランナブルの開始時に排他エリアへの入場を行い，ランナブルの終了時に排他エリアからの退場を行う． ・BSWM エンティティ(BSW スケジューラブル)の場合 BSWM 開発者が，システム設計時に BSW スケジューラブル実行中は排他エリア内での動作となることを定義し， SCHM に保証させる方法．SCHM は，BSW スケジューラブルの開始時に排他エリアへの入場を行い，BSW スケジューラブルの終了時に排他エリアからの退場を行う． ・BSWM エンティティ(BSW スケジューラブル以外)の場合 SCHM は，対象 BSWM エンティティに対する実行時入退場を行わない【rte_sws_7524】.

2.12.2.2 排他エリア実現メカニズム

RTEGEN は，排他エリアの実現のために以下の排他エリア実現メカニズムを提供する．どの排他エリア実現メカニズムを適用するかは，コンフィギュレーションにより選択できる．

表 2-28 排他エリア実現メカニズム一覧

排他エリア実現 メカニズム	内容
全割込みのブロック	全割込みを無効化する，もしくは，サスペンドすることで実現する． 本メカニズムは，非常に短時間の競合の回避のために有用である． ただし，システム全体のタイミングや高優先度の OS タスクにまで影響を及ぼすことに注意が必要である．
OS 割込みのブロック	OS 割込みをサスペンドすることで実現する． 本メカニズムは，非常に短時間の競合の回避のために有用である． ただし，システム全体のタイミングや高優先度の OS タスクにまで影響を及ぼすことに注意が必要である．
OS リソースの獲得	OS タスク間で OS リソースを獲得し合うことで実現する． 本メカニズムには，割込みブロックに比べ，一貫性保証を必要とする OS タスク間にのみ影響が集約されるという利点がある． 本メカニズムは，コア間の排他に使用できない点に注意が必要である．
OS スピンロックの獲得	OS タスク間で OS スピンロックを獲得し合うことで実現する． 本メカニズムには，OS リソースの獲得に比べ，コア間の排他に使用できるという利点がある．
排他なし	排他しない． 排他エリアを設定しても，本メカニズムを指定することで，排他しないことができる．本メカニズムは，ECU インテグレーション時に排他が不要となった場合に有用である．

2.12.3 排他エリアの操作

2.12.3.1 排他エリアへの入場

ランナブルの排他エリア入退場方法が「明示的入退場」である場合，RTE は，排他エリア毎に，排他エリアへの入場のための API(Rte_Enter)を提供する【rte_sws_3515】．排他エリアを保持する SW-C は，提供された API を使用することで，排他エリアへの入場を行うことができる．

ランナブルの排他エリア入退場方法が「エクスキュータブル実行時入退場」である場合，RTE は，ランナブルが開始する際に排他エリアへの入場を自動的に行う【rte_sws_a_0031】．

BSWM エンティティの排他エリア入退場方法が「明示的入退場」である場合，SCHM は，排他エリア毎に，排他エリアへの入退場のための API(SchM_Enter)を提供する【rte_sws_7523】.

BSW スケジューラブルの排他エリア入退場方法が「エクスキュータブル実行時入退場」である場合，SCHM は，BSW スケジューラブルが開始する際に排他エリアへの入場を自動的に行う【rte_sws_a_0061】.

排他エリアへの入場の挙動

Rte_Enter/SchM_Enter，もしくは「エクスキュータブル実行時入退場」により排他エリアへの入場が開始された際の挙動は，排他メカニズムにより異なる．各排他エリア実現メカニズムにおける，排他エリアへの入場が開始された場合の挙動は以下の通りである．

データ一貫性保証メカニズム	排他エリアへの入場が開始された場合の挙動
全割込みのブロック	全割込みの禁止を開始する【rte_sws_3504】.
OS 割込みのブロック	OS 割込みの禁止を開始する【rte_sws_5164】.
OS リソースの獲得	OS リソースを獲得する【rte_sws_3595】.
OS スピンロックの獲得	OS スピンロックを獲得する【nrte_sws_0032】.
排他なし	何も行わない【nrte_sws_0275】.

サポート範囲の制限

排他エリアへのアクセスが競合する可能性がないことが自明な場合には，RTEGEN は，排他エリア実現メカニズムを適用せず，排他エリアへの入場時は何も行わない実装を提供する【rte_sws_3504】【rte_sws_5164】【rte_sws_3595】．しかし，A-RTEGEN では，これらの仕様を制限としてサポートしない【irte_sws_0001】.

2.12.3.2 排他エリアからの退場

ランナブルの排他エリア入退場方法が「明示的入退場」である場合，RTE は，排他エリア毎に，排他エリアからの退場のための API(Rte_Exit)を提供する【rte_sws_3515】．排他エリアを保持する SW-C は，提供された API を使用することで，排他エリアからの退場を行うことができる.

ランナブルの排他エリア入退場方法が「エクスキュータブル実行時入退場」である場合，RTE は，ランナブルが終了する際に，排他エリアから退場を自動的に行う【rte_sws_a_0032】.

BSWM エンティティの排他エリア入退場方法が「明示的入退場」である場合，SCHM は，排他エリア毎に，排他エリアからの退場のための API(SchM_Exit)を提供する【rte_sws_7523】.

BSW スケジューラブルの排他エリア入退場方法が「エクスキュータブル実行時入退場」である場合，SCHM は，BSW スケジューラブルが終了する際に，排他エリアから退場を自動的に行う【rte_sws_a_0062】.

排他エリアからの退場の挙動

Rte_Exit/SchM_Exit，もしくは「エクスキュータブル実行時入退場」により排他エリアからの退場が開始された際の挙動は，排他エリア実現メカニズムにより異なる．各排他エリア実現メカニズムにおける，排他エリアからの退場時の挙動は以下の通りである.

データ一貫性保証 メカニズム	排他エリアからの退場時の挙動
全割込みのブロック	全割込みの禁止を終了する【rte_sws_3504】.
OS 割込みのブロック	OS 割込みの禁止を終了する【rte_sws_5164】.
OS リソースの獲得	OS リソースを解放する【rte_sws_3595】.
OS スピンロックの獲得	OS スピンロックを解放する【nrte_sws_0033】.
排他なし	何も行わない【nrte_sws_0276】.

排他エリアへのアクセスが競合しない場合の挙動

排他エリアへのアクセスが競合する可能性がないことが自明な場合には，RTEGEN は，排他エリア実現メカニズムを適用せず，排他エリアからの退場時には何も行わない実装を提供する【rte_sws_3504】【rte_sws_5164】【rte_sws_3595】．しかし，A-RTEGEN では，これらの仕様を制限としてサポートしない〔irte_sws_0001〕．

2.12.4 排他エリアの設定

2.12.4.1 排他エリア

排他エリアは，*排他エリア(ExclusiveArea)*により指定する．

2.12.4.2 排他エリア入退場方法

*エクスキュータブル(ExecutableEntity)の明示的排他エリア入場(canEnterExclusiveArea)*が排他エリアを参照する場合，そのエクスキュータブルに対し，排他エリア入退場方法「明示的入退場」を有効とする【rte_sws_a_0033】．

*エクスキュータブル(ExecutableEntity)の排他エリア内実行(runsInsideExclusiveArea)*が排他エリアを参照する場合，そのエクスキュータブルに対し，排他エリア入退場方法「エクスキュータブル実行時入退場」を有効とする【rte_sws_a_0034】．

2.12.4.3 排他エリア実現メカニズム

排他エリアに適用する排他エリア実現メカニズムは，排他エリアを参照する**排他エリア実現メカニズム(RteExclusiveAreaImplMechanism)**により指定する．設定値に応じて，適用する排他エリア実現メカニズムは以下の通り【rte_sws_a_0035】．

表 2-29 設定値と適用する排他エリア実現メカニズム

設定値	適用する排他エリア実現メカニズム
ALL_INTERRUPT_BLOCKING	全割込みのブロック【rte_sws_3504】.
OS_INTERRUPT_BLOCKING	OS 割込みのブロック【rte_sws_5164】.
OS_RESOURCE	OS リソースの獲得【rte_sws_3595】.
OS_SPINLOCK	OS スピンロックの獲得【nrte_sws_0034】.
NONE	排他なし【nrte_sws_0276】.

使用 OS リソースの指定

排他エリア実現メカニズム「OS リソースの獲得」で使用する OS リソースは，**排他エリア実現メカニズム(RteExclusiveAreaImplMechanism)**の**使用 OS リソース(RteExclusiveAreaOsResourceRef)**により指定する【rte_sws_a_0036】.

使用 OS スピンロックの指定

排他エリア実現メカニズム「OS スピンロックの獲得」で使用する OS スピンロックは，**排他エリア実現メカニズム(RteExclusiveAreaImplMechanism)**の**使用 OS スピンロック(RteExclusiveAreaOsSpinlockRef)**により指定する【rte_sws_a_0037】.

2.13 ライフサイクル管理

RTE/SCHM の起動，および停止は，EcuStateManager により行われる(図 2-26)． このとき，RTE の起動は，SCHM 起動後に実施されなければならない【rte_sws_ext_7577】．また，RTE の停止は，SCHM 停止前に実施されなければならない【rte_sws_ext_7576】.

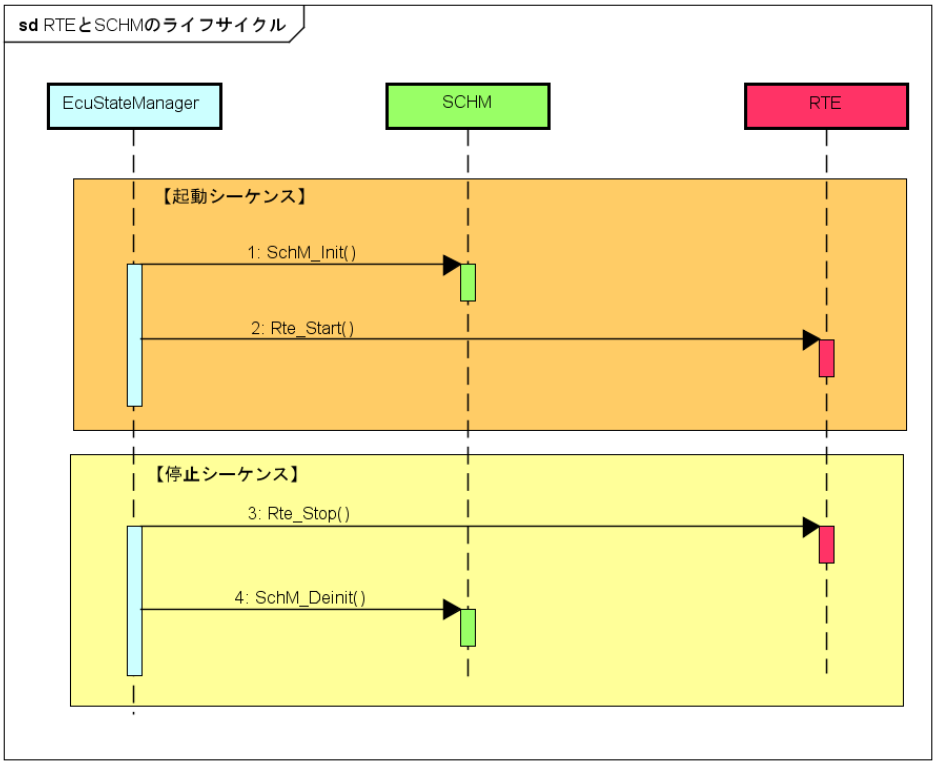

図 2-26 RTE と SCHM のライフサイクル

RTE/SCHM は，EcuStateManager に対して，RTE/SCHM を起動，および停止するための機能を提供する．また RTE は，停止，もしくは再起動中のパーティション内の SW-C が動作しないように動作を抑制する機能を提供する．SCHM は，停止中に BSWM が動作しないように動作を抑制する機能を提供する．

2.13.1 SCHM ライフサイクルの状態

2.13.1.1 SCHM の状態遷移

SCHM の状態遷移を以下に定義する．SCHM の状態マシンは，コア毎に存在する．

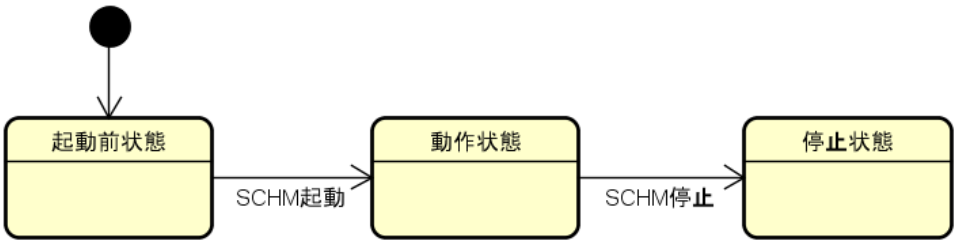

図 2-27 SCHM の状態遷移図

表 2-30 SCHM の状態

SCHM の状態	説明
起動前状態	SCHM が起動されておらず，EcuStateManager からの起動待ちである.
動作状態	SCHM が動作中である.
停止状態	SCHM が停止された.

表 2-31 SCHM の遷移

SCHM の遷移	説明（箇条書きはアクションの内容）
SCHM 起動	EcuStateManager から SCHM に対して開始要求があった. ・　2.13.2.1 の SCHM 起動の挙動に記載する処理を実施する.
SCHM 停止	EcuStateManager から SCHM に対して停止要求があった. ・　2.13.2.2 の SCHM 停止の挙動に記載する処理を実施する.

表 2-32 SCHM の状態遷移表

遷移 SCHM の状態	SCHM 起動	SCHM 停止
起動前状態（初期状態）	動作状態	－
動作状態	－	停止状態
停止状態	－	－

2.13.2 SCHM ライフサイクルの操作

2.13.2.1 SCHM 起動

SCHM は，SCHM 起動のための API(SchM_Init)を提供する．EcuStateManager は，提供された API を使用して SCHM を起動することができる.

本 SCHM は，SCHM 起動の際，SCHM の依存する BSWM の初期化を行わない【nrte_sws_0214】. 本 SCHM の依存する BSWM の初期化の実施は，BSWM，および EcuStateManager の責任である.

SCHM は，SCHM 起動をコア毎に独立して行う【nrte_sws_0215】．そのため，EcuStateManager は，コア毎に SchM_Init を呼び出す必要がある.

SCHM 起動の挙動

SchM_Init により，呼出し元コアの SCHM 起動が開始された際の挙動は以下の通りである【nrte_sws_0216】．SCHM 起動はブロックせず，有限の実行時間で復帰する.

(1) SchM_Init の呼出し元のコアを動作状態に移行する.
(2) SchM_Init の呼出し元のコアにマッピングされている全てのモードマシンインスタンスを初期モードへ遷移させる〔rte_sws_7532〕.
(3) SchM_Init の呼出し元のコアにマッピングされている周期イベントとバックグラウンドイベントによって起動する BSW スケジューラブルの起動を開始する〔rte_sws_7574〕〔rte_sws_7584〕.

SCHM 起動の制限

SCHM 起動は，以下の BSWM の起動後に行わなければならない【nrte_sws_ext_0024】.

・ OS

SCHM 起動は，「起動前状態」にいる間に行なわれなければならない【nrte_sws_ext_0026】.

2.13.2.2 SCHM 停止

SCHM は，SCHM 停止のための API(SchM_Deinit)を提供する．EcuStateManager は，提供された API を使用して SCHM を停止することができる.

本 SCHM は，SCHM 停止の際，SCHM の依存する BSWM の終了処理を行わない【nrte_sws_0217】. 本 SCHM の依存する BSWM の終了処理の実施は，BSWM，および EcuStateManager の責任である.

SCHM は，SCHM 停止をコア毎に独立して行う【nrte_sws_0218】．そのため，EcuStateManager は，コア毎に SchM_Deinit を呼び出す必要がある.

SCHM 停止の挙動

SchM_Deinit により，呼出し元コアの SCHM 停止が開始された際の挙動は以下の通りである【nrte_sws_0219】．SCHM 停止はブロックせず，有限の実行時間で復帰する.

(1) SchM_Deinit の呼出し元のコアを停止状態に移行する.

SCHM 停止の制限

SCHM 停止は，以下の BSWM の終了処理の前に行わなければならない【nrte_sws_ext_0027】.

・ OS

SCHM 停止は,「起動前状態」以外の状態にいる間に行なわれなければならない【nrte_sws_ext_0028】.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，SCHM が停止状態である場合，SCHM は，そのコアの BSW スケジューラブルの起動要求，および開始要求を無視し，要求がないものとして扱うと規定している【rte_sws_7586】. しかし，この仕様を厳密に守るとオーバヘッドが大きいため，本 SCHM ではサポートしない【nrte_sws_0220】.

2.13.2.3 BSWM の初期化と終了

本 SCHM は，BSWM の初期化，および終了のための機能を提供しない【nrte_sws_0221】.

2.13.3 RTE ライフサイクルの状態

2.13.3.1 RTE の状態遷移

RTE の状態遷移を以下に定義する．RTE の状態マシンは，コア毎に存在する.

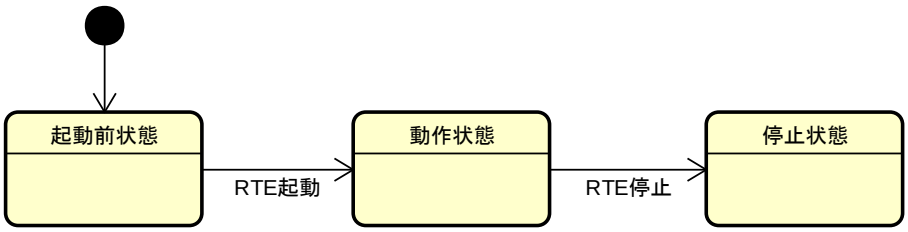

図 2-28 RTE の状態遷移図

表 2-33 RTE の状態

RTE の状態	説明
起動前状態	RTE が起動されておらず，EcuStateManager からの起動待ちである.
動作状態	RTE が動作中である.
停止状態	RTE が停止された.

表 2-34 RTE の遷移

RTE の遷移	説明（箇条書きはアクションの内容）
RTE 起動	EcuStateManager から RTE に対して開始要求があった. ・　2.13.4.1 の RTE 起動の挙動に記載する処理を実施する.
RTE 停止	EcuStateManager から RTE に対して停止要求があった. ・　2.13.4.2 の RTE 停止の挙動に記載する処理を実施する.

表 2-35 RTE の状態遷移表

遷移 RTE の状態	RTE 起動	RTE 停止
起動前状態（初期状態）	動作状態	－
動作状態	－	停止状態
停止状態	－	－

2.13.3.2 パーティションの状態遷移

パーティションの状態遷移を以下に定義する.

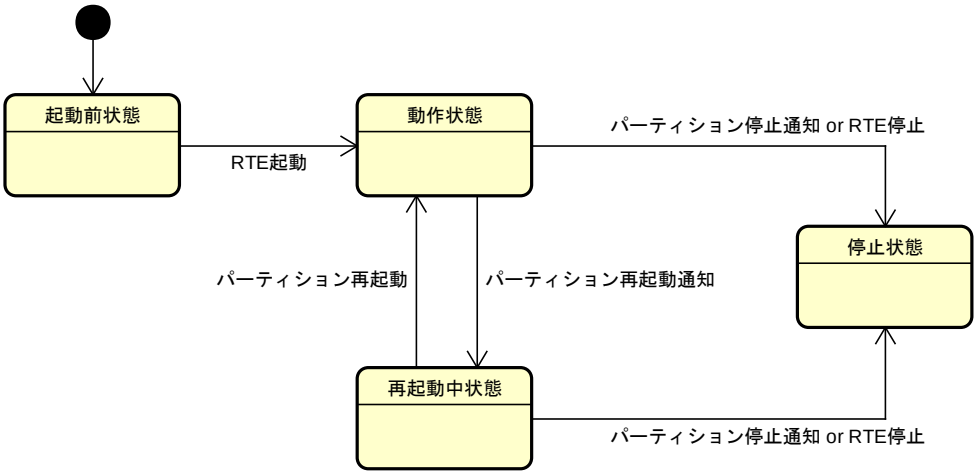

図 2-29 パーティションの状態遷移図

表 2-36 パーティションの状態

パーティションの状態	説明
起動前状態	パーティションが起動されておらず，RTE の起動待ちである.
動作状態	パーティションが動作中である.
再起動中状態	パーティションが再起動中である.
停止状態	パーティションが停止された.

表 2-37 パーティションの遷移

パーティションの遷移	説明（箇条書きはアクションの内容）
パーティション停止通知	ECU インテグレーションコードからパーティションに対して，パーティションの停止が通知された. ・　2.13.4.3 のパーティション停止通知の挙動に示す処理を実施する.
パーティション再起動通知	ECU インテグレーションコードからパーティションに対して，パーティションの再起動が通知された. ・　2.13.4.3 のパーティション再起動通知の挙動に示す処理を実施する.
パーティション再起動	ECU インテグレーションコードからパーティションに対して，パーティションの再起動要求があった. ・　2.13.4.4 のパーティション再起動の挙動に示す処理を実施する.

表 2-38 パーティションの状態遷移

パーティションの遷移 状態	RTE 起動	RTE 停止	パーティション停止通知	パーティション再起動通知	パーティション再起動
起動前状態 (初期状態)	動作状態	－	－	－	－
動作状態	－	停止状態	停止状態	再起動中状態	－
再起動中状態	－	停止状態	停止状態	－	動作状態
停止状態	－	－	－	－	－

2.13.4 RTE ライフサイクルの操作

2.13.4.1 RTE 起動

RTE は，RTE 起動のための API(Rte_Start)を提供する．EcuStateManager は，提供された API を使用して RTE を起動することができる.

本 RTE は，RTE 起動の際，RTE の依存する BSWM の初期化を行わない【nrte_sws_0035】．本 RTE の依存する BSWM の初期化の実施は，BSWM，および ECU インテグレーションコードの責任である．

RTE は，RTE 起動をコア毎に独立して行う【rte_sws_7615】．そのため，EcuStateManager は，コア毎に Rte_Start を呼び出す必要がある．

RTE 起動の挙動

Rte_Start により，呼出し元コアの RTE 起動が開始された際の挙動は以下の通りである【nrte_sws_0077】．RTE 起動はブロックせず，有限の実行時間で復帰する．

(1) Rte_Start の呼出し元のコアにマッピングされている全てのパーティションについて，パーティションに所属する全ての SW-C の以下の状態を初期状態とする【rte_sws_7616】〔rte_sws_a_0038〕．
  - RTE 起動時に初期化するよう設定された全データ要素について，受信したデータ要素の値，および状態

(2) Rte_Start の呼出し元のコアにマッピングされている全パーティションを動作状態に移行する．

(3) Rte_Start の呼出し元のコアを動作状態に移行する．

(4) Rte_Start の呼出し元のコアにマッピングされている周期イベントとバックグラウンドイベントを起動契機とするランナブルの起動を開始する〔rte_sws_7575〕〔rte_sws_7178〕．

RTE 起動の制限

RTE 起動は，以下の BSWM の起動後に行わなければならない【nrte_sws_ext_0007】．

- OS
- COM

マスタコアの RTE 起動は，どのスレーブコアの RTE 起動よりも先に行わなければならない【nrte_sws_ext_0008】．

RTE 起動は，「起動前状態」にいる間に行なわれなければならない【nrte_sws_ext_0009】.

2.13.4.2 RTE 停止

RTE は，RTE 停止のための API(Rte_Stop)を提供する．EcuStateManager は，提供された API を使用して RTE を停止することができる．

本 RTE は，RTE 停止の際，RTE の依存する BSWM の終了処理を行わない【nrte_sws_0037】．本 RTE の依存する BSWM の終了処理の実施は，BSWM，および ECU インテグレーションコードの責任である．

RTE は，RTE 停止をコア毎に独立して行う【rte_sws_a_0040】．そのため，EcuStateManager は，コア毎に Rte_Stop を呼び出す必要がある．

RTE 停止の挙動

Rte_Stop により，呼出し元コアの RTE 停止が開始された際の挙動は以下の通りである【nrte_sws_0078】．RTE 停止はブロックせず，有限の実行時間で復帰する．

(2) Rte_Stop の呼出し元のコアを停止状態に移行する．

(3) Rte_Stop の呼出し元のコアに所属する全パーティションを停止状態に移行する．

RTE 停止の制限

RTE 停止は，以下の BSWM の終了処理の前に行わなければならない【nrte_sws_ext_0010】．

- OS
- COM

マスタコアの RTE 停止は，全てのスレーブコアの RTE 停止の後に行わなければならない【nrte_sws_ext_0011】．

RTE 停止は，「起動前状態」以外のいずれかの状態にいる間に行なわれなければならない【nrte_sws_ext_0012】.

RTE が停止状態の場合の挙動

RTE が停止状態である場合，RTE は，そのコアのランナブルの起動要求，および開始要求を無視し，要求がないものとして扱う【rte_sws_2538】．

AUTOSAR 仕様との違い

AUTOSAR 仕様では，RTE が停止状態である場合，RTE は，C/S 連携のオペレーション呼出し要求，および S/R 連携からのデータ送受信要求を無視し，要求がなかったものとして扱うと規定されている【rte_sws_2535】【rte_sws_2536】．しかし，これらの仕様を厳密に守るとオーバヘッドが大きいため，本 RTE では，S/R 連携のデータ送受信におけるデータ一貫性のみ保証する【nrte_sws_0055】．

2.13.4.3 パーティションの停止，および再起動の通知

RTE は，パーティション毎に，以下の API を提供する．

- パーティションが停止することを RTE に通知する API(Rte_PartitionTerminated)
- パーティションが再起動することを RTE に通知するための API(Rte_PartitionRestarting)

ECU インテグレーションコードは，提供された API を使用して，パーティションが停止，もしくは再起動することを RTE に通知する．RTE は，パーティションが停止，および再起動したことの通知を受け取って，パーティションの状態遷移を行うのみであり，パーティションにマッピングされた OSAP を停止，および再起動する機能を提供しない．

パーティション停止通知の挙動

Rte_PartitionTerminated により，パーティション停止通知が開始された場合の挙動は以下の通りである【nrte_sws_0079】．

(1) 停止対象のパーティションの状態を停止状態に移行する.
(2) 停止対象のパーティションが送信側となり，データのタイムアウト監視が有効なパーティション間連携先がある場合，受信側パーティションの受信データセットの状態を「タイムアウト状態」に移行する〔rte_sws_2710〕．その後，データ受信エラーイベントを起動契機とするランナブルを起動する〔rte_sws_a_0068〕．ただし，A-RTE は，データ受信エラーイベントを起動契機とするランナブルを起動しない〔irte_sws_0028〕.

パーティション再起動通知の挙動

Rte_PartitionRestarting により，パーティション再起動通知が開始された場合の挙動は以下の通りである【nrte_sws_0080】.

(1) 再起動対象のパーティションの状態を再起動中状態に移行する.
(2) 停止対象のパーティションが送信側となり，データのタイムアウト監視が有効なパーティション間連携先がある場合，受信側パーティションの受信データセットの状態を「タイムアウト状態」に移行する〔rte_sws_2710〕．その後，データ受信エラーイベントを起動契機とするランナブルを起動する〔rte_sws_a_0068〕．ただし，A-RTE は，データ受信エラーイベントを起動契機とするランナブルを起動しない〔irte_sws_0028〕.

パーティションが停止状態，もしくは再起動中状態の場合の挙動

RTE は，パーティションが停止状態，もしくは再起動中状態である場合，そのパーティションに所属するエクスキュータブル実行インスタンスの起動，および開始要求を無視し，要求がないものとして扱う【rte_sws_7604】.

2.13.4.4 パーティションの再起動

RTE は，パーティションを起動するための API(Rte_RestartPartition)を提供する．ECU インテグレーションコードは，提供された API を使用して，パーティションを再起動することができる.

パーティションの再起動の挙動

Rte_RestartPartition により，パーティションの再起動が開始された際の挙動は以下の通りである【nrte_sws_0081】.

(1) パーティションに所属する全ての SW-C を初期状態とする【rte_sws_2735】．初期化対象とする状態は以下の通りである【rte_sws_a_0038】.
    - パーティションの再起動時に初期化するよう設定された全データ要素について，受信したデータ要素の値，および状態〔rte_sws_2517〕.
(2) パーティションを動作状態に移行する.

2.13.4.5 SW-C の初期化と終了

本 RTE は，SW-C の初期化，および終了のための機能を提供しない【nrte_sws_0082】.

ECU インテグレーションコードは，RTE 起動，もしくはパーティションの再起動時に，各 SW-C の初期化を行う必要がある．また，ECU インテグレーションコードは，RTE 停止，もしくはパーティションの停止時に，各 SW-C の終了処理を行う必要がある．

ECU インテグレーションコードによる，SW-C の初期化，および終了処理内容の例

各 SW-C に初期化のためのランナブル，および終了のためのランナブルを定義し，初期化，および終了処理の定義を行う．

- システム起動時には，Rte_Start 後に，各 SW-C の初期化のためのランナブルを実行する．
- システム終了時には，Rte_Stop 後に，各 SW-C の終了のためのランナブルを実行する．
- パーティションの再起動時には，リスタートタスクから，各 SW-C の終了のためのランナブルを実行後，Rte_RestartPartition を呼び出し，Rte_RestartPartition の呼出し完了後に，各 SW-C の初期化のためのランナブルを実行する．

2.13.5 ライフサイクルの実現方式

2.13.5.1 SCHM の状態管理

本 SCHM の機能仕様範囲では，EcuStateManager のライフサイクル管理により，SCHM のライフサイクル管理が実現可能である．このため，本 SCHM では，SCHM の状態管理を行わない【nrte_sws_0222】．

2.13.5.2 RTE の状態管理

本 RTE の機能仕様範囲では，EcuStateManager, および OS のライフサイクル管理により，RTE のライフサイクル管理が実現可能である．このため，本 RTE では，RTE の状態管理を行わない【nrte_sws_0052】．

2.13.5.3 パーティションの停止，および再起動

RTE は，パーティションの停止，および再起動機能を，OS の提供する OSAP を使用して実現する【rte_sws_a_0043】．

2.13.5.4　排他方式

RTE 内部で RTE ライフサイクルを実現する場合の排他方式は実装定義とする【nrte_sws_0271】.
A-RTE の排他方式は以下の通りである【irte_sws_0021】.

RTE ライフサイクル API	排他方式
Rte_Start	排他しない. (本 RTE は，Rte_Start 呼び出し中は，ランナブルが起動していないことを期待している【nrte_sws_ext_0032】)
Rte_Stop	排他しない. (Rte_Stop では，RTE 内部バッファの更新をしない)
Rte_PartitionTerminated	〔irte_sws_0014〕および，〔irte_sws_0020〕に従う.
Rte_PartitionRestarting	〔irte_sws_0014〕および，〔irte_sws_0020〕に従う.
Rte_RestartPartition	〔irte_sws_0014〕および，〔irte_sws_0020〕に従う.

2.14 ファイル構成

Rte.c，および Rte.h を除いて，RTE，および RTE/SCHM 共有のファイル名は接頭辞を"Rte_"とする【rte_sws_7139】．SCHM 固有のファイル名の接頭辞は”SchM_”とする【rte_sws_7288】. RTE/SCHM の生成ファイルであるか否かは，接頭辞"Rte_"または”SchM_”により識別できる．RTE/SCHM のファイル構成を表 2-39 に示す.

表 2-39 ファイル構成

モジュール	種別	ファイル名	参照
RTE/SCHM	RTE ヘッダ	Rte.h	2.14.1
RTE/SCHM	ライフサイクルヘッダ	Rte_Main.h	2.14.2
RTE	アプリケーションヘッダ	Rte_<swcnp>.h	2.14.3
RTE/SCHM	RTE タイプヘッダ	Rte_Type.h	2.14.4
RTE	アプリケーションタイプヘッダ	Rte_<swcnp>_Type.h	2.14.5
RTE	VFB トレースヘッダ	Rte_Hook.h	2.14.6
RTE	RTE コンフィギュレーションヘッダ	Rte_Cfg.h	2.14.7
SCHM	モジュール連結タイプヘッダ	SchM_<bsnp>_Type.h	2.14.8
SCHM	モジュール連結ヘッダ	SchM_<bsnp>.h	2.14.9
RTE/SCHM	RTE ソース	Rte.c	2.14.10
RTE/SCHM		Rte_Partition_<Partition>.c	
RTE		Rte_Cbk.h	

ここで,

・<swcnp>：SW-C 型のショートネーム

・<bsnp>： BSWM ディスクリプションのショートネーム【rte_sws_7593】

AUTOSAR 仕様との違い

AUTOSAR 仕様では，*BSWM エンティティ(BswModuleEntity)*に *SCHM 名接頭辞(BswScheduleNamePrefix)*を指定している場合，<bsnp>は *SCHM 名接頭辞(BswSchedulerNamePrefix)*のショートネームとしている【rte_sws_7594】.

しかし，A-RTEGEN では，*SCHM 名接頭辞(BswSchedulerNamePrefix)*はサポートしない【irte_sws_0008】.

2.14.1 RTE ヘッダ

RTE ヘッダは，ECU 毎に生成する必要のない RTE の固定のコードを定義する.

2.14.1.1 ファイル名

RTE ヘッダを定義するファイル名は Rte.h である【rte_sws_1157】.

2.14.1.2 ファイル内容

RTE ヘッダは Std_Types.h をインクルードする【rte_sws_1164】.

RTE ヘッダの内容は本仕様書で規定した要素に限定せず，RTE ヘッダにどのようなコードを含むかは，実装定義とする【nrte_sws_0039】.

2.14.2 ライフサイクルヘッダ

ライフサイクルヘッダは RTE ライフサイクル API, および SCHM ライフサイクル API を定義する. RTE ライフサイクル API の詳細は，3.9 節を参照．SCHM ライフサイクル API の詳細は，3.12 節を参照.

2.14.2.1 ファイル名

ライフサイクルヘッダを定義するファイル名は Rte_Main.h である【rte_sws_1158】.

2.14.2.2 ファイル内容

ライフサイクルヘッダは RTE ヘッダをインクルードする【rte_sws_1159】.

2.14.3 アプリケーションヘッダ

アプリケーションヘッダは，RTE を使用する SW-C が必要とする RTE API，および関連するデータ構造を定義する.

RTEGEN は，コンフィギュレーションで定義された SW-C 型毎にアプリケーションヘッダを生成する【rte_sws_1000】.

アプリケーションヘッダはメモリオブジェクトを作成するコードを含んではならない【rte_sws_3786】.

2.14.3.1 ファイル名

アプリケーションヘッダを定義するファイル名は，"Rte_"の後に SW-C 型のショートネームが続き，最後に".h"を付与したものとする【rte_sws_1003】.

2.14.3.2 スコープ

アプリケーションヘッダは，その SW-C に関する情報のみを含む【rte_sws_1004】.

アプリケーションヘッダの情報を制限することで，アプリケーションヘッダを使用する SW-C が，その SW-C のための API，およびデータ構造以外にアクセスしないことをコンパイル時に保証する.

2.14.3.3 ファイル内容

多重インクルード防止マクロ

同じモジュールに複数のアプリケーションヘッダをインクルードしてはならない．そのため，アプリケーションヘッダは最初に以下の宣言を含む【rte_sws_1006】．

```
#ifdef RTE_APPLICATION_HEADER_FILE
#error Multiple application header files included.
#endif /* RTE_APPLICATION_HEADER_FILE */
#define RTE_APPLICATION_HEADER_FILE
```

アプリケーション固有型のインクルード

アプリケーションヘッダはアプリケーションタイプヘッダをインクルードする【rte_sws_7131】．

C/C++サポート

アプリケーションヘッダは C/C++ ソースの両方で有効とする【rte_sws_1005】．

アプリケーションヘッダ内の全ての定義は以下の記述の後に記述する【rte_sws_3709】．

```
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
```

アプリケーションヘッダ内の全ての定義は以下の記述の前に記述する【rte_sws_3710】．

```
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
```

初期値

アプリケーションヘッダはデータの初期値定数を含む．

RTE-SW-C 間インタフェース

アプリケーションヘッダは以下の RTE-SW-C 間のインタフェース定義を含む．

- SW-C から使用可能な RTE API の API マッピング
- ランナブルのエントリポイント関数のプロトタイプ宣言

SW-C 型から使用可能な RTE API のみを，その SW-C のアプリケーションヘッダにより指定する【rte_sws_1276】．

2.14.4 RTE タイプヘッダ

RTE タイプヘッダは，コンフィギュレーションにより可変の，RTE 固有の型宣言を定義する．

RTE タイプヘッダで宣言される型は，RTE の内部データの実装や RTE API で使用される．

RTEGEN は，実装データ型を定義する RTE タイプヘッダを生成する【rte_sws_1160】．

このヘッダはジェネレーションフェーズで生成される．

実装データ型

実装レベルでのデータ型．C 言語において，typedef により定義される型に相当する．

2.14.4.1 ファイル名

RTE タイプヘッダを定義するファイル名は Rte_Type.h とする【rte_sws_1161】．

2.14.4.2 ファイル内容

RTE タイプヘッダは，RTE が使用するかどうかに関係なく，〔rte_sws_7104〕，〔rte_sws_7110〕，〔rte_sws_7114〕，〔rte_sws_7144〕，〔rte_sws_7109〕，〔rte_sws_7148〕に従ったデータ型の型宣言を含む【rte_sws_2648】．RTE が使用しないデータ型は，SW-C で使用される．

RTE タイプヘッダは，RTE ヘッダをインクルードする【rte_sws_1163】．

2.14.4.3 C/C++

RTE タイプヘッダ内ではデータ型は typedef を用いて宣言する【rte_sws_1162】．

2.14.5 アプリケーションタイプヘッダ

アプリケーションタイプヘッダは SW-C 固有の列挙リテラル，および範囲データ値を定義する．

このヘッダはジェネレーションフェーズで生成する．

RTEGEN は，SW-C 型毎にアプリケーションタイプヘッダを生成する【rte_sws_7120】．

アプリケーションタイプヘッダはメモリオブジェクトを作成するコードを含んではならない【rte_sws_7121】．

2.14.5.1 ファイル名

アプリケーションタイプヘッダを定義するファイル名は，接頭辞"Rte_"の後に SW-C 型のショートネームが続き，末尾に"_Type.h"を付与したものとする【rte_sws_7122】．

2.14.5.2 スコープ

アプリケーションタイプヘッダは，その SW-C に関連する情報のみを含む【rte_sws_7123】．

アプリケーションタイプヘッダは C/C++ソースの両方に有効とする【rte_sws_7124】．

アプリケーションタイプヘッダ内の全ての定義は以下の記述の後に記述する【rte_sws_7125】．

```
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
```

アプリケーションタイプヘッダ内の全ての定義は以下の記述の前に記述する【rte_sws_7126】．

```
#ifdef __cplusplus
```

```
} /* extern "C" */
#endif /* __cplusplus */
```

アプリケーションタイプヘッダは多重インクルードを防止するために以下の記述を含む【rte_sws_7678】.

```
#ifndef RTE_<SW-C>_TYPE_H
#define RTE_<SW-C>_TYPE_H
 …
/*
 * Contents of file
 */
 …
#endif /* !RTE_<SW-C>_TYPE_H */
```

ここで，<SW-C>は SW-C 型のショートネームを表す.

2.14.5.3 ファイル内容

アプリケーションヘッダとは異なり，アプリケーションタイプヘッダは，同一モジュール内に複数インクルードすることをサポートする．複数のアプリケーションヘッダのインクルードは，ある BSWM が複数の AUTOSAR サービスを使用するような場合に必要となる.

アプリケーションタイプヘッダは RTE タイプヘッダをインクルードする【rte_sws_7127】.

列挙データ型

アプリケーションタイプヘッダは列挙定数の定義を含む.

範囲データ型

アプリケーションタイプヘッダは上限値，および下限値定数の定数定義を含む.

2.14.6 VFB トレースヘッダ

VFB トレースヘッダは VFB トレースイベントを定義する.

本 RTE では VFB トレースをサポートしないが，互換性のために空のファイルを生成する.

VFB トレースヘッダはジェネレーションフェーズのみで生成する【rte_sws_1319】.

VFB トレースヘッダを定義するファイル名は Rte_Hook.h とする【rte_sws_1250】.

2.14.7 RTE コンフィギュレーションヘッダ

RTE コンフィギュレーションヘッダは RTE の振る舞いに影響を与えるユーザ定義を記述する.

本 RTE では VFB トレースをサポートしないが，互換性のために空のファイルを生成する.

VFB トレースヘッダを使用するときは，このヘッダをコンパイラのインクルードパスに配置しなければならない．このヘッダは RTEGEN が生成する.

2.14.7.1 ファイル名

RTE コンフィギュレーションヘッダを定義するファイル名は Rte_Cfg.h とする【rte_sws_1321】.

2.14.7.2 ファイル内容

RTE コンフィギュレーションヘッダは Std_Types.h をインクルードする【rte_sws_7641】.

2.14.8 モジュール連結タイプヘッダ

RTEGEN は，入力された *BSW 実装(BswImplementation)*が参照する *BSW 内部振る舞い(BswInternalBehavior)*のモジュール連結タイプヘッダを生成する【nrte_sws_0234】.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，*BSW 内部振る舞い(BswInternalBehavior)*に登録されている *SCHM 名接頭辞(BswScheduleNamePrefix)*毎にモジュール連結タイプヘッダを生成すると規定している【rte_sws_7503】.

しかし，A-RTEGEN は，*SCHM 名接頭辞(BswSchedulerNamePrefix)*はサポートしない〔irte_sws_0008〕.

2.14.8.1 ファイル名

モジュール連結タイプヘッダファイル名は，以下の通りである【rte_sws_7295】.

SchM_<bsnp>_Type.h

ここで,

・<bsnp> は，BSWM ディスクリプションのショートネーム〔rte_sws_7593〕.

AUTOSAR 仕様との違い

AUTOSAR 仕様では BSWM の *ベンダ ID(vendorId)*と *ベンダ API 接中辞(vendorApiInfix)*を接頭辞に含めている【rte_sws_7295】.

しかし, A-RTEGEN では, SCHM のマルチインスタンスはサポートしないため, *ベンダ ID(vendorId)*と *ベンダ API 接中辞(vendorApiInfix)*は使用しない【irte_sws_0009】.

2.14.8.2 スコープ

モジュール連結タイプヘッダは，BSWM に関連するデータ型のみを含む【rte_sws_7296】.

モジュール連結タイプヘッダは，C/C++ソースの両方に有効とする【rte_sws_7297】.

モジュール連結タイプヘッダ内の全ての定義は以下の記述の後に記述する【rte_sws_7298】.

```
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
```

モジュール連結タイプヘッダ内の全ての定義は以下の記述の前に記述する【rte_sws_7299】.

```
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
```

2.14.8.3 ファイル内容

モジュール連結タイプヘッダは RTE タイプヘッダをインクルードする【rte_sws_7500】.

2.14.9 モジュール連結ヘッダ

RTEGEN は，入力された *BSW 実装(BswImplementation)*が参照する *BSW 内部振る舞い(BswInternalBehavior)*のモジュール連結ヘッダを生成する【nrte_sws_0235】.

SCHM API を使用する，または，BSW スケジューラブルを実装する BSWM ファイルは，モジュール連結ヘッダファイルをインクルードしなければならない【rte_sws_ext_7512】.

モジュール連結ヘッダはメモリオブジェクトを作成するコードを含んではならない【rte_sws_7502】.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，*BSW 内部振る舞い(BswInternalBehavior)*に登録されている *SCHM 名接頭辞(BswScheduleNamePrefix)*毎にモジュール連結ヘッダを生成すると規定している【rte_sws_7501】.

しかし，A-RTEGEN は，*SCHM 名接頭辞(BswSchedulerNamePrefix)*はサポートしない〔irte_sws_0008〕.

2.14.9.1　ファイル名

モジュール連結タイプヘッダファイル名は，以下の通りである【rte_sws_7504】.

SchM_<bsnp>.h

ここで,

・<bsnp> は，BSWM ディスクリプションのショートネーム〔rte_sws_7593〕.

AUTOSAR 仕様との違い

AUTOSAR 仕様では BSWM の *ベンダ ID(vendorId)*と *ベンダ API 接中辞(vendorApiInfix)*を接頭辞に含めている【rte_sws_7504】.

しかし, A-RTEGEN では, SCHM のマルチインスタンスはサポートしないため, *ベンダ ID(vendorId)*と *ベンダ API 接中辞(vendorApiInfix)*は使用しない〔irte_sws_0009〕.

2.14.9.2　スコープ

モジュール連結ヘッダは，BSWM に関係する宣言のみを含む【rte_sws_7505】.

2.14.9.3 ファイル内容

モジュール連結ヘッダはモジュール連結タイプヘッダをインクルードする【rte_sws_7506】.

モジュール連結ヘッダで定義する SCHM API は，BSWM に関連するもののみが定義されていなければならない【rte_sws_7510】.

BSWM がオブジェクトコードで配布される場合，SCHM API は関数で実装されなければならない【rte_sws_7511】.

C/C++サポート

モジュール連結ヘッダは，C/C++ソースの両方に有効とする【rte_sws_7507】.

モジュール連結ヘッダ内の全ての定義は以下の記述の後に記述する【rte_sws_7508】.

```
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */
```

モジュール連結ヘッダ内の全ての定義は以下の記述の前に記述する【rte_sws_7509】.

```
#ifdef __cplusplus
} /* extern "C" */
#endif /* __cplusplus */
```

2.14.10RTE ソース

RTE ソースは，RTE，および SCHM 機能の実装を定義する.

2.14.10.1使用ヘッダ

互換モードでは，生成された RTE/SCHM は Os.h をインクルードする【rte_sws_1257】.

互換モードでは，生成された RTE は Com.h をインクルードする【rte_sws_3794】.

互換モードでは，生成された RTE/SCHM は Rte.h をインクルードする【rte_sws_1279】.

互換モードでは，生成された RTE は VFB トレースヘッダをインクルードする【rte_sws_1326】.

エントリポイント関数の宣言〔rte_sws_7194〕を除いて，RTE/SCHM は AUTOSAR メモリマップメカニズムを使用するために，MemMap.h を使ってメモリオブジェクトをマッピングする【rte_sws_3788】.

本 RTE は，IOC を使用する場合，Ioc.h をインクルードする【nrte_sws_0236】.

2.14.10.2C/C++

ECU 内の全てのコアで共有される RTE/SCHM コードを含む C モジュールの名前は Rte.c とする【rte_sws_1169】.

RTEGEN は，Rte.c の他に，RTE/SCHM をいくつかのファイルに分割してもよい【rte_sws_7140】. パーティション構成の ECU において，本 RTEGEN は，パーティション毎に，パーティション固有のコードを定義する Rte_Partition_<Partition>.c を生成する【nrte_sws_0183】. ここで, <Partition>は,

パーティションのショートネームである．前述したソースファイルに加えて，RTE/SCHM をいくつのファイルに分割するかは，実装定義とする【nrte_sws_0040】．

Rte.c，および Rte_Partition_<Partition>.c 間でシンボルの再定義は行ってはならない【nrte_sws_0184】.

パーティションの権限に応じて OS コードを切り替えできるように，本 RTE/SCHM は，Rte_Partition_<Partition>.c の Os.h のインクルードの前に，OS の要求する信頼，非信頼を示すマクロ〔NOS1157〕を定義する【nrte_sws_0187】． A-RTEGEN では，ATK2 の要求する以下のマクロを定義する【irte_sws_0006】.

パーティションの権限	定義するマクロ名
信頼	TOPPERS_TRUSTED
非信頼	TOPPERS_NON_TRUSTED

AUTOSAR 仕様との違い

AUTOSAR 仕様では，マルチコア構成の ECU において，RTEGEN は，コア毎に，コア固有のコードを定義するソースファイルを生成すると規定されていた【rte_sws_2712】.しかし，本 RTEGEN では，OS のメモリ保護機能のパーティションへの適用を容易とするため，この仕様を削除し，パーティション毎に，パーティション固有のコードを定義するソースファイルを生成する仕様を追加した〔nrte_sws_0183〕.

AUTOSAR 仕様では，RTE/SCHM は，Rte.c 内で定義されたグローバル変数，および静的変数のみを使用する制約が規定されていた【rte_sws_2711】.しかし，本 RTE/SCHM では，パーティション固有のコード〔nrte_sws_0183〕においてグローバル変数の定義を許容するため，この制約を削除した.

2.14.10.3 ファイル内容

本来，RTE/SCHM の内容はベンダ固有である．API，データ型定義のように RTE/SCHM の外部に公開する項目のみを仕様として定義する.

生成 API

RTEGEN は SW-C が RTE API を呼び出したときに起動される生成関数を定義する【rte_sws_1266】. 生成関数の詳細は，3.1.5 節を参照.

コールバック

RTE API に対する生成関数に加えて，RTE はデータの受信時等に COM が実行するコールバックを含む.

RTEGEN は，適切な COM シグナル/COM シグナルグループに対する COM コールバック関数を定義する【rte_sws_1264】.

RTEGEN は，COM コールバック関数のプロトタイプを含むヘッダを定義するファイルを別途生成する【rte_sws_3795】.

コールバック関数のプロトタイプを含むヘッダを定義するファイル名は，Rte_Cbk.h とする【rte_sws_3796】.

タスクボディ

RTE/SCHM は，互換モードのときだけ RTEGEN が生成した OS タスクに対するタスクボディを定義する.

互換モードでは， RTEGEN は，全ての RTE/SCHM のタスクボディを定義する【rte_sws_1277】【rte_sws_a_0063】.

RTE ライフサイクル API

RTEGEN は，RTE ライフサイクル API を定義する【rte_sws_1197】. RTE ライフサイクル API の詳細は，3.9 節を参照.

SCHM ライフサイクル API

RTEGEN は，SCHM ライフサイクル API を定義する. SCHM ライフサイクル API の詳細は，3.12 節を参照.

2.14.10.4 リエントラント性

並行実行が想定されるような RTE/SCHM のコードは全てリエントラントでなければならない【nrte_sws_0083】.

並行実行が想定されないようなコードに対しては，この条件は適用されない.

2.15 コンフィギュレーション違反チェック

RTEGEN は，RTE/SCHM コードの生成前に，以下の制約内容に合致しているかコンフィギュレーション情報を検証し，違反が検出された場合は，全ての違反箇所に対するメッセージを表示して RTE/SCHM コードの生成処理を中断する【nrte_sws_0087】.

仕様番号	制約内容
【nrte_sws_0047】	**RTE(Rte)**が定義されていなければならない.
【nrte_sws_0101】	*実装データ型(ImplementationDataType)*の*カテゴリ(category)*が *VALUE* の場合には，*ソフトウェアベース型(baseType)*が定義されていなければならない.
【nrte_sws_0102】	*ECU 設定値コレクション(EcucValueCollection)*が 1 つだけ定義されていなければならない.
【nrte_sws_0104】	*SWC 内部振る舞い(SwcInternalBehavior)*の*排他エリア(ExclusiveArea)*を参照する**排他エリア実現メカニズム(RteExclusiveAreaImplementation)**が定義されている場合,その数は 1 つでなければならない.
【nrte_sws_0105】	**排他エリア実現メカニズム(RteExclusiveAreaImplementation)**の**排他エリア実現メカニズム種別(RteExclusiveAreaImplMechanism)**が OS_RESOURCE のとき，**使用 OS リソース(RteExclusiveAreaOsResourceRef)**が定義されていなければならない.
【nrte_sws_0106】	**排他エリア実現メカニズム(RteExclusiveAreaImplementation)**の**排他エリア実現メカニズム種別(RteExclusiveAreaImplMechanism)**が OS_SPINLOCK のとき，**使用 OS スピンロック(RteExclusiveAreaOsSpinlockRef)**が定義されていなければならない.
【nrte_sws_0107】	**コンフィグ対象 RTE イベント(RteEventRef)**が*オペレーション呼出しイベント(OperationInvokedEvent)*のとき，**マッピング先 OS タスク(RteMappedToTaskRef)**が定義されてはならない.
【nrte_sws_0109】	**コンフィグ対象 RTE イベント(RteEventRef)**が*周期イベント(TimingEvent)*のとき，**マッピング先 OS タスク(RteMappedToTaskRef)**が定義されていなければならない．（RTEGEN は，直接関数呼び出しの場合を除き，*ランナブル(RunnableEntity)*を起動する *RTE イベント(RTEEvent)*が OS タスクに関連付けされていない設定を拒否する【rte_sws_2204】と，RTEGEN は，*ランナブル(RunnableEntity)*と，*BSW スケジューラブル(BswSchedulableEntity)*の OS タスクやタスクボディへの関連付け情報が不足した設定を拒否する【rte_sws_2254】を上位仕様とする）
【nrte_sws_0110】	**コンフィグ対象 RTE イベント(RteEventRef)**が*周期イベント(TimingEvent)*のとき，**使用 OS アラーム(RteUsedOsAlarmRef)**が定義されていなければならない.
【nrte_sws_0111】	RTE コードを生成する場合，*システム(System)*の*ルートソフトウェアコンポジション(rootSoftwareComposition)*が定義されていなければならない.
【nrte_sws_0112】	*RTE イベント(RteEvent)*を参照する **RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**が 1 つだけ定義されていなければならない.
【nrte_sws_0113】	*RTE イベント(RteEvent)*の*イベント起動対象ランナブル(startOnEvent)*が定義されていなければならない.

仕様番号	制約内容
【nrte_sws_0114】	SW-C から使用される *アプリケーションデータ型(ApplicationDataType)*は，1 つの *実装データ型(ImplementationDataType)*とマッピングされなければならない．（RTEGEN は，RTE 生成に影響する *アプリケーションデータ型(ApplicationDataType)*が *実装データ型(ImplementationDataType)*とマッピングされていない設定を拒否する【rte_sws_7028】を上位仕様とする）
【nrte_sws_0115】	*コンポジション SW-C 型(CompositionSwComponentType)*の *所属コンポーネント(component)*は 1 つ以上定義されていなければならない．
【nrte_sws_0116】	パーティション構成の場合，コア毎に，BSWM 配置パーティションが 1 つだけ定義されていなければならない．
【nrte_sws_0117】	データの無効化が有効である場合，*無効値(invalidValue)*が定義されていなければならない．
【nrte_sws_0118】	**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**の**使用 OS アラーム(RteUsedOsAlarmRef)**により参照される *OS アラーム(OsAlarm)*に対して，**OS アクティベーション設定(RteUsedOsActivation)**が 1 つだけ定義されていなければならない．
【nrte_sws_0119】	同一の *データ要素(dataElement)*が複数の *要求側連携仕様(requiredComSpec)*から参照されてはならない．
【nrte_sws_0120】	同一の *データ要素(dataElement)*が複数の *提供側連携仕様(providedComSpec)*から参照されてはならない．
【nrte_sws_0121】	パーティション毎に，そのパーティションを参照する *OSAP(OsApplication)*が 1 つだけ定義されていなければならない．
【nrte_sws_0122】	*データフィルタ種別(dataFilterType)*が maskedNewEqualsX のとき，*データフィルタ(dataFilter)*の mask と x が定義されていなければならない．
【nrte_sws_0123】	*データフィルタ種別(dataFilterType)*が maskedNewMaskedOld のとき，*データフィルタ(dataFilter)*の mask が定義されていなければならない．
【nrte_sws_0124】	*データフィルタ種別(dataFilterType)*が maskedNewDiffersX のとき，*データフィルタ(dataFilter)*の mask と x が定義されていなければならない．
【nrte_sws_0125】	*データフィルタ種別(dataFilterType)*が newIsWithin のとき，*データフィルタ(dataFilter)*の max と min が定義されていなければならない．
【nrte_sws_0126】	*データフィルタ種別(dataFilterType)*が newIsOutside のとき，*データフィルタ(dataFilter)*の max と min が定義されていなければならない．
【nrte_sws_0127】	*データフィルタ種別(dataFilterType)*が oneEveryN のとき，*データフィルタ(dataFilter)*の period と offset が定義されていなければならない．
【nrte_sws_0128】	*データセマンティクス受信側連携仕様(NonqueuedSenderComSpec)*と *データセマンティクス受信側連携仕様(NonqueuedReceiverComSpec)*は，*データ要素*

仕様番号	制約内容
	*(dataElement)*の*型(AutosarDataType)*に対して適切な*初期値(initValue)*を持たなければならない.
【nrte_sws_0129】	マルチコア構成である場合，全パーティションにコアが割り当てられていなければならない.
【nrte_sws_0130】	サーバオペレーションに対し，1 つの*オペレーション呼出しイベント(OperationInvokedEvent)*が定義されていなければならない.
【nrte_sws_0131】	*OS タスク(OsTask)*を参照するいずれかの **RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**で**使用 OS イベント(RteUsedOsEventRef)**が定義されている場合，*OS タスク(OsTask)*を参照する全ての **RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**には，**使用 OS イベント(RteUsedOsEventRef)**が定義されていなければならない.
【nrte_sws_0132】	*変数データプロトタイプ(VariableDataPrototype)*は，複数の*無効化ポリシ(InvalidationPolicy)*から参照されてはならない.
【nrte_sws_0133】	**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**の**起動オフセット(RteActivationOffset)**は，**OS アクティベーション設定(RteUsedOsActivation)**の**起動オフセット期待値(RteExpectedActivationOffset)**以上でなければならない.
【nrte_sws_0134】	**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**の**起動オフセット(RteActivationOffset)**と **OS アクティベーション設定(RteUsedOsActivation)**の**起動オフセット期待値(RteExpectedActivationOffset)**が異なる場合，2 つの値の差は，**OS アクティベーション設定(RteUsedOsActivation)**の**起動周期期待値(RteExpectedTickDuration)**の倍数でなければならない.
【nrte_sws_0135】	*周期イベント(TimingEvent)*の*起動周期(period)*は，**OS アクティベーション設定(RteUsedOsActivation)**の**起動周期期待値(RteExpectedTickDuration)**の倍数でなければならない.
【nrte_sws_0136】	パーティション構成の場合，*ルートソフトウェアコンポジションプロトタイプ(RootSwCompositionPrototype)*に定義された全ての *SW-C プロトタイプ(SwComponentPrototype)*は，*パーティション(EcucPartition)*にマッピングされていなければならない.
【nrte_sws_0137】	データセマンティクスの場合，*初期値(initValue)*が定義されていなければならない.
【nrte_sws_0138】	*アトミック SW-C 型(AtomicSwComponentType)*は，*内部振る舞い(internalBehavior)*が定義されていなければならない.（RTEGEN は，*アトミック SW-C 型(AtomicSwComponentType)*が*内部振る舞い(SwcInternalBehavior)*を含まない設定を拒否する【rte_sws_7686】を上位仕様とする）

仕様番号	制約内容
【nrte_sws_0139】	*ポートプロトタイプ(PortPrototype)*は，複数の*ポート API オプション(PortAPIOption)*から参照されてはならない.
【nrte_sws_0140】	送信側データ要素の無効化が有効である場合，全受信側データ要素に*無効値(invalidValue)*が存在しなければならない.
【nrte_sws_0141】	**排他エリア実現メカニズム(RteExclusiveAreaImplementation)**の**排他エリア実現メカニズム手段(RteExclusiveAreaImplMechanism)**に COOPERATIVE_RUNNABLE_PLACEMENT が設定されてはならない.
【nrte_sws_0142】	*データセマンティクス受信側連携仕様(NonqueuedReceiverComSpec)*は，*データセマンティクスの変数データプロトタイプ(VariableDataPrototype)*を参照していなくてはならない.
【nrte_sws_0143】	*イベントセマンティクス受信側連携仕様(QueuedReceiverComSpec)*は，*イベントセマンティクスの変数データプロトタイプ(VariableDataPrototype)*を参照していなくてはならない.
【nrte_sws_0144】	*データセマンティクス送信側連携仕様(NonqueuedSenderComSpec)*は，*データセマンティクスの変数データプロトタイプ(VariableDataPrototype)*を参照していなくてはならない.
【nrte_sws_0145】	*無効化ポリシ(InvalidationPolicy)*はデータセマンティクスの*変数データプロトタイプ(VariableDataPrototype)*を参照していなくてはならない.
【nrte_sws_0147】	あるタスクに複数のランナブルがマッピングされている場合，それぞれの**ランナブル実行順番(RtePositionInTask)**は定義され，かつ，ユニークでなければならない.
【nrte_sws_0149】	マルチコア構成である場合，マスタコアの BSWM 配置パーティションが存在しなくてはならない.
【nrte_sws_0150】	シングルコア構成である場合，パーティションはコアに割り付けられてはならない.
【nrte_sws_0152】	*OS(Os)*が定義されていなければならない.
【nrte_sws_0153】	*無効値(invalidValue)*は，以下のいずれかでなくてはならない. ・ NumericalValueSpecification ・ TextualValueSpecification ・ (以上のいずれかを参照する)ConstantReference
【nrte_sws_0154】	パーティション構成の場合，**RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**により参照される全ての *OS タスク(OsTask)*は，1 つの*パーティション(EcucPartition)*にマッピングされていなければならない.
【nrte_sws_0155】	*実装データ型(ImplementationDataType)*の*ベース型(baseType)*の*ベース型サイズ(baseTypeSize)*が定義されていなければならない.

仕様番号	制約内容
【nrte_sws_0156】	同一のショートネームを持つ*カテゴリ(category)*が *VALUE* の*実装データ型(ImplementationDataType)*が複数存在する場合，それらの*ベース型(baseType)*の*ネイティブ宣言(nativeDeclaration)*は一致していなければならない.
【nrte_sws_0157】	ある SW-C において，*C/S インタフェース(ClientServerInterface)*，およびアプリケーションエラー*(ApplicationError)*の 2 つのショートネームが一致するアプリケーションエラー*(ApplicationError)*が複数使用されている場合，それらのエラーコード*(errorCode)*は一致してなければならない.
【nrte_sws_0189】	*BSW 実装(BswImplementation)*が指す *BSWM ディスクリプション(BswModuleDescription)*の *BSW 内部振る舞い(BswInternalBehavior)*の数は 1 つだけ定義されていなければならない.
【nrte_sws_0190】	*OS タスク(OsTask)*を参照する **RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**が存在する場合，対象 *OS タスク(OsTask)*を参照する **BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**が存在してはならない.
【nrte_sws_0191】	*OS タスク(OsTask)*を参照する **BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**が存在する場合，対象 *OS タスク(OsTask)*を参照する **RTE イベント-OS タスクマッピング(RteEventToTaskMapping)**が存在してはならない.
【nrte_sws_0192】	*BSW スケジューラブル(BswSchedulableEntity)*により参照される *BSW モジュールエントリ(BswModuleEntry)*の *callType* は *scheduled* でなければならい.
【nrte_sws_0193】	*BSW スケジューラブル(BswSchedulableEntity)*により参照される *BSW モジュールエントリ(BswModuleEntry)*の *executionContext* は *task* でなければならない.
【nrte_sws_0194】	コンフィグ対象 **BSW イベント(RteBswEventRef)**が*周期イベント(BswTimingEvent)*のとき，**マッピング先 OS タスク(RteBswMappedToTaskRef)**が定義されていなければならない.（RTEGEN は，直接関数呼び出しの場合を除き，*BSW スケジューラブル(BswSchedulableEntity)*を起動する *BSW イベント(BswEvent)*が OS タスクに関連付けされていない設定を拒否する【rte_sws_7516】と〔rte_sws_2254〕を上位仕様とする）
【nrte_sws_0195】	コンフィグ対象 *BSW イベント(RteBswEventRef)*が*周期イベント(BswTimingEvent)*のとき，**使用 OS アラーム(RteBswUsedOsAlarmRef)**が定義されていなければならない.
【nrte_sws_0196】	*BSW イベント(BswEvent)*を参照する **BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**が 1 つだけ定義されていなければならない.
【nrte_sws_0197】	**BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**の**使用 OS**

仕様番号	制約内容
	アラーム(RteBswUsedOsAlarmRef)により参照される *OS アラーム(OsAlarm)* に対して，**OS アクティベーション設定(RteUsedOsActivation)**が 1 つだけ定義されていなければならない.
【nrte_sws_0198】	*OS タスク(OsTask)*を参照するいずれかの **BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**で**使用 OS イベント(RteBswUsedOsEventRef)**が定義されている場合，*OS タスク(OsTask)*を参照する全ての **BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**には，**使用 OS イベント(RteBswUsedOsEventRef)**が定義されていなければならない.
【nrte_sws_0199】	**BSWイベント-OSタスクマッピング(RteBswEventToTaskMapping)**の**起動オフセット(RteBswActivationOffset)**は，**OS アクティベーション設定(RteUsedOsActivation)**の**起動オフセット期待値(RteExpectedActivationOffset)**以上でなければならない.
【nrte_sws_0200】	**BSWイベント-OSタスクマッピング(RteBswEventToTaskMapping)**の**起動オフセット(RteBswActivationOffset)**と**OS アクティベーション設定(RteUsedOsActivation)**の**起動オフセット期待値(RteExpectedActivationOffset)**が異なる場合，2 つの値の差は，**OS アクティベーション設定(RteUsedOsActivation)**の**起動周期期待値(RteExpectedTickDuration)**の倍数でなければならない.
【nrte_sws_0201】	*周期イベント(BswTimingEvent)の起動周期(period)*は，**OS アクティベーション設定(RteUsedOsActivation)**の**起動周期期待値(RteExpectedTickDuration)**の倍数でなければならない.
【nrte_sws_0202】	あるタスクに複数の BSW スケジューラブル(BswSchedulableEntity)がマッピングされている場合，それぞれの **BSW スケジューラブル実行順番(RteBswPositionInTask)**は定義され，かつ，ユニークでなければならない.
【nrte_sws_0203】	パーティション構成の場合，**BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**により参照される全ての *OS タスク(OsTask)*は，1 つの*パーティション(EcucPartition)*にマッピングされていなければならない.
【nrte_sws_0204】	パーティション構成の場合，**BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)**により参照される全ての *OS タスク(OsTask)*が所属する*パーティション(EcucPartition)*の *BSWM 実行パーティション(EcucPartitionBswModuleExecution)*は true でなければならない.
【nrte_sws_0205】	パーティション構成の場合，*排他エリア(ExclusiveArea)*を参照する *BSW スケジューラブル(BswSchedulableEntity)*の所属*パーティション(EcucPartition)*は全て同じでなければならない.
【nrte_sws_0226】	SCHM コードを生成する場合，*BSW 実装(BswImplementation)*が定義されてい

仕様番号	制約内容
	なければならない.
【nrte_sws_0227】	*BSW 内部振る舞い(BswInternalBehavior)*の*排他エリア(ExclusiveArea)*を参照する **BSW 排他エリア実現メカニズム(RteBswExclusiveAreaImpl)**が定義されている場合,その数は 1 つでなければならない.
【nrte_sws_0228】	**BSW 排他エリア実現メカニズム(RteBswExclusiveAreaImpl)**の**排他エリア実現メカニズム種別(RteExclusiveAreaImplMechanism)**が OS_RESOURCE のとき，**使用 OS リソース(RteBswExclusiveAreaOsResourceRef)**が定義されていなければならない.
【nrte_sws_0229】	**BSW 排他エリア実現メカニズム(RteBswExclusiveAreaImpl)**の**排他エリア実現メカニズム種別(RteExclusiveAreaImplMechanism)**が OS_SPINLOCK のとき，**使用 OS スピンロック(RteBswExclusiveAreaOsSpinlockRef)**が定義されていなければならない.
【nrte_sws_0230】	**BSW 排他エリア実現メカニズム(RteBswExclusiveAreaImpl)**の**排他エリア実現メカニズム手段(RteExclusiveAreaImplMechanism)**に COOPERATIVE_RUNNABLE_PLACEMENT が設定されてはならない.
【nrte_sws_0231】	*BSW 内部振る舞い(BswInternalBehavior)*の *BSWM エンティティ(BswModuleEntity)*が *BSW スケジューラブル(BswSchedulableEntity)*でない場合，*排他エリア内実行(runsInsideExclusiveArea)*は定義されてはならない.
【nrte_sws_0232】	*BSW 実装(BswImplementation)*の *BSW 内部振る舞い(BswInternalBehavior)*を参照する **BSWM インスタンス(RteBswModuleInstance)**は 1 つだけ定義されていなければならない.
【nrte_sws_0240】	*カテゴリ(category)*が ARRAY *の実装データ型(ImplementationDataType)*の*実装データ型要素(ImplementationDataTypeElement)*は 1 つだけ定義されていなければならない.
【nrte_sws_0242】	*カテゴリ(category)*が ARRAY *の実装データ型(ImplementationDataType)*の*実装データ型要素(ImplementationDataTypeElement)* はプリミティブ実装データ型でなければならない.
【nrte_sws_0244】	*カテゴリ(category)* が STRUCTURE *の実装データ型(ImplementationDataType)* の *実装データ型要素(ImplementationDataTypeElement)* はプリミティブ実装データ型でなければならない.
【nrte_sws_0246】	*カテゴリ(category)*が UNION *の実装データ型(ImplementationDataType)*の*実装データ型要素(ImplementationDataTypeElement)* はプリミティブ実装データ型でなければならない.
【nrte_sws_0247】	*実装データ型要素(ImplementationDataTypeElement)*の*カテゴリ(category)*が

仕様番号	制約内容
	VALUE の場合には，*ソフトウェアベース型(baseType)*が定義されていなければならない.
【nrte_sws_0255】	*ポインタ実装データ型のターゲットカテゴリ(targetCategory)*は *VALUE*，または *TYPE_REFERENCE* でなければならない.
【nrte_sws_0256】	*ポインタ実装データ型のターゲットカテゴリ(targetCategory)*が *VALUE* の場合，参照するベース型*(baseType)*の*ネイティブ宣言(nativeDeclaration)*が定義されていなければならない.
【nrte_sws_0257】	*ポインタ実装データ型のターゲットカテゴリ(targetCategory)* が *TYPE_REFERENCE* の場合，参照する*実装データ型(ImplementationDataType)*が定義されていなければならない.
【nrte_sws_0277】	*COM シグナルグループ(ComSignalGroup)* の *comSystemTemplateSignalGroup.iSignalGroup* が参照する *iSignal* の集合と，*COM シグナルグループ*が持つそれぞれの *COM グループシグナル*の *comSystemTemplateSignal.iSignal* の集合は一致しなければならない.（複合データ型要素の ECU 間連携において，データマッピング要素は，プリミティブ要素の AUTOSAR シグナル毎に，少なくとも 1 つの *COM シグナル(ComSignal)* を含まなければならない【rte_sws_4508】. を上位仕様とする）
【nrte_sws_0278】	*ISignalGroup* が参照する *iSignal.systemSignal* の集合と，*ISignalGroup* が参照する *systemSignalGroup.systemSignal* の集合は一致しなければならない.（①同じ複合データ型の要素と関連付く各シグナルは，同じシグナルグループに関連付く．②2 つのシグナルが同じ複合データ型の要素と関連付かない場合，それらは同じシグナルグループに関連付かない．③シグナルが複合データ型の要素に関連付かない場合，そのシグナルはシグナルグループに関連付かない．【rte_sws_2557】を上位仕様とする）
【nrte_sws_0279】	*S/R 連携シグナルグループマッピング(SenderReceiverToSignalGroupMapping)* の参照する *signalGroup.systemSignal* の数は，*dataElement* を通信するために必要なサイズでなければならない.
【nrte_sws_0280】	*S/R 連携シグナルグループマッピング(SenderReceiverToSignalGroupMapping)* の参照する *signalGroup.systemSignal* の集合と，*SenderReceiverToSignalGroupMapping* の持つ *typeMapping* 配下で参照される *systemSignal* の集合は一致しなければならない.（〔rte_sws_2557〕を上位仕様とする）
【nrte_sws_0281】	*S/R 連携シグナルグループマッピング(SenderReceiverToSignalGroupMapping)* が *S/R 連携配列型マッピング(SenderRecArrayTypeMapping)*を持つ場合，配下の *arrayElementMapping.indexedArrayElement* は対応する*実装データ型*

仕様番号	制約内容
	(ImplementationDataType) の *実装データ型要素 (ImplementationDataTypeElement)*を参照し，必要な *index* を網羅していなければならない.
【nrte_sws_0282】	*S/R 連携シグナルグループマッピング(SenderReceiverToSignalGroupMapping)* が *S/R 連携レコード型マッピング(SenderRecRecordTypeMapping)*を持つ場合，配下の *recordElementMapping* は対応する *実装データ型 (ImplementationDataType)* の通信対象となる *実装データ型要素 (ImplementationDataTypeElement)*を漏れなく参照していなければならない.
【nrte_sws_0283】	*COM シグナル(ComSignal)*の持つ *comSystemTemplateSystemSignal* が参照する *iSignal* は定義され，*iSignalGroup* は未定義でなくてはならない.（〔rte_sws_2557〕を上位仕様とする）
【nrte_sws_0284】	C/S 連携では，*要求側ポートプロトタイプ(RPortPrototype)*を，別のコアに配置される SW-C の*提供側ポートプロトタイプ(PPortPrototype)*に接続してはならない.
【nrte_sws_0285】	*S/R 連携シグナルマッピング*(S*enderReceiverToSignalMapping)*は，*データ要素参照(dataElementIref)*として，*プリミティブデータ型*か uint8 型の配列を参照しなければならない.
【nrte_sws_0286】	C/S 連携では，*要求側ポートプロトタイプ(RPortPrototype)*を，別の非信頼パーティションに配置される SW-C の*提供側ポートプロトタイプ(PPortPrototype)*に接続してはならない.
【nrte_sws_0287】	ECU 間連携に使用する *S/R インタフェース(SenderReceiverInterface)*のデータ*要素(dataElement)*を，ポインタ実装データ型で定義してはならない.
【nrte_sws_0288】	C/S 連携で，*引数 (ArgumentDataPrototype)* の *実装データ型 (ImplementationDataType)*のカテゴリ*(category)*が *DATA_REFERENCE* の場合，*方向(direction)*は *IN* でなければならない.
【nrte_sws_0289】	*ポートインタフェース (PortInterface)* 内で参照する *実装データ型 (ImplementationDataType)*のカテゴリ*(category)*が *DATA_REFERENCE* の場合，送信/受信コンポーネントは *ServiceSwComponentType*，*ComplexDeviceDriverSwComponentType*, *EcuAbstractionSwComponentType* のいずれかでなければならない.
【nrte_sws_0290】	*S/R インタフェース(SenderReceiverInterface)*のデータ要素*(dataElement)*の型*(type)*としてプリミティブ実装データ型もしくはプリミティブ実装データ型の再定義型が設定されている場合を除き，*データセマンティクス受信側連携仕様 (NonqueuedReceiverComSpec)*に*フィルタ(dataFilter)*が設定されていてはならない.

仕様番号	制約内容
【nrte_sws_0291】	*S/R インタフェース(SenderReceiverInterface)*の*データ要素(dataElement)*の*型(type)*としてプリミティブ実装データ型もしくはプリミティブ実装データ型の再定義型が設定されている場合を除き，*モデル違反無効化ポリシ(InvalidationPolicy)*の*無効値受信時処理(handleInvalid)*が *keep*，もしくは *replace* に設定されていてはならない.
【nrte_sws_0292】	*SWC 内部振る舞い(SwcInternalBehavior)*の*直接ランナブル変数(explicitRunnableVariable)*の*変数データプロトタイプ(VariableDataPrototype)*は，*初期値(InitValue)*と*型(ImplementationDataType)*を持ち，初期値は型に対して適切でなければならない
【nrte_sws_0293】	*ポート定義引数値(PortDefinedArgumentValue)*の*値(value)*は，*型(valueType)*に対して適切でなければならない
【nrte_sws_0294】	*SWC 内部振る舞い(SwcInternalBehavior)*の*直接ランナブル変数(explicitRunnableVariable)*の*変数データプロトタイプ(VariableDataPrototype)*は，*ランナブル(RunnableEntity)*の*書き込みローカル変数(writtenLocalVariable)*もしくは*読み込みローカル変数(readLocalVariable)*の*変数アクセス(VariableAccess)*から参照されなければならない
【nrte_sws_0295】	*カテゴリ(category)*が *ARRAY* の*実装データ型(ImplementationDataType)*の*実装データ型要素(ImplementationDataTypeElement)*の *arraySize* は 1 以上でなければならない.
【nrte_sws_0298】	*カテゴリ(category)* が *STRUCTURE* の*実装データ型(ImplementationDataType)* の*実装データ型要素(ImplementationDataTypeElement)* は 1 つ以上でなければならない.
【nrte_sws_0299】	*OsIocInitValue* は，*実装データ型(ImplementationDataType)*の C 言語による初期値設定が可能なフォーマットでなければならない.
【nrte_sws_0300】	*ランナブル(RunnableEntity)*の*書き込みローカル変数(writtenLocalVariable)*の*変数アクセス(VariableAccess)*の参照先は，親である *SWC 内部振る舞い(SwcInternalBehavior)*の*直接ランナブル変数(explicitRunnableVariable)*である*変数データプロトタイプ(VariableDataPrototype)*でなければならない.
【nrte_sws_0301】	*ランナブル(RunnableEntity)*の*読み込みローカル変数(readLocalVariable)*の*変数アクセス(VariableAccess)*の参照先は，親である *SWC 内部振る舞い(SwcInternalBehavior)*の*直接ランナブル変数(explicitRunnableVariable)*である*変数データプロトタイプ(VariableDataPrototype)*でなければならない.
【nrte_sws_0302】	*カテゴリ(category)*が *UNION* の*実装データ型(ImplementationDataType)*の*実装データ型要素(ImplementationDataTypeElement)* は 2 つ以上でなければならない.

仕様番号	制約内容
【nrte_sws_0303】	同一のショートネームを持つ*カテゴリ(category)*が *ARRAY* の*実装データ型(ImplementationDataType)*が複数存在する場合，*実装データ型要素(ImplementationDataTypeElement)*の*ベース型(baseType)*の*ネイティブ宣言(nativeDeclaration)*が一致するか，*実装データ型要素*の*実装データ型*のショートネームが一致していなければならない.
【nrte_sws_0304】	同一のショートネームを持つ*カテゴリ(category)*が *STRUCTURE* の*実装データ型(ImplementationDataType)*が複数存在する場合，保持する全ての*実装データ型要素(ImplementationDataTypeElement)*の*ベース型(baseType)*の*ネイティブ宣言(nativeDeclaration)*が一致するか，*実装データ型要素*の*実装データ型*のショートネームが一致していなければならない.
【nrte_sws_0305】	同一のショートネームを持つ*カテゴリ(category)*が *UNION* の*実装データ型(ImplementationDataType)*が複数存在する場合，保持する全ての*実装データ型要素(ImplementationDataTypeElement)*の*ベース型(baseType)*の*ネイティブ宣言(nativeDeclaration)*が一致するか，*実装データ型要素*の*実装データ型*のショートネームが一致していなければならない.
【nrte_sws_0306】	*同一のショートネームを持つカテゴリ(category)*が *TYPE_REFERENCE* の*実装データ型(ImplementationDataType)*が複数存在する場合，*実装データ型要素(ImplementationDataTypeElement)*から参照される*実装データ型*の*ショートネーム*が一致，かつ，*実装データ型*が互換していなければならない.
【nrte_sws_0307】	同一の*ショートネーム*を持つ*カテゴリ(category)*が *DATA_REFERENCE* の*実装データ型(ImplementationDataType)*が複数存在する場合，*実装データ型要素(ImplementationDataTypeElement)*から参照される*実装データ型*の*ショートネーム*が一致しなければならない.
【nrte_sws_0328】	1 つの*モードユーザ*に対して，モード切替通知を行う複数の*モードマネージャ*があってはならない（RTEGEN は，同じ受信者に対する複数の送信者からのモード切替通知をサポートしない【rte_sws_2670】を上位仕様とする）
【nrte_sws_0329】	*BSW モードマネージャポリシー(BswModeSenderPolicy)*の*キュー長(queueLength)*は 1 以上でなければならない
【nrte_sws_0330】	1 つの*モード宣言グループプロトタイプ (ModeDeclarationGroupPrototype)* で使用される*モード宣言グループ (ModeDeclarationGroup)* を参照する*モード要求型マップ (ModeRequestTypeMap)* は 1 つでなければならない
【nrte_sws_0331】	モード切替連携は，パーティション内連携でなければならない
【nrte_sws_0337】	BSW 提供モードグループ参照(RteBswProvidedModeGroupRef)の参照先は，*提供モードグループ(providedModeGroup)*でければならない
【nrte_sws_0338】	BSW 要求モードグループ参照(RteBswRequiredModeGroupRef)の参照先は，*要*

仕様番号	制約内容
	*求モードグループ(requiredModeGroup)*でなければならない
【nrte_sws_0339】	BSW 要求モードグループ接続(RteBswRequiredModeGroupConnection)の BSW 提供モードグループ参照(RteBswProvidedModeGroupRef)の参照先と BSW 要求モードグループ参照(RteBswRequiredModeGroupRef)の参照先には，互換性がなければならない
【nrte_sws_0340】	*モード要求型マップ (ModeRequestTypeMap) の実装データ型 (implementationDataType)*は，*モード宣言 (ModeDeclaration)*数を表現可能な符号なし整数型でなければならない
【nrte_sws_0341】	*BSWM エンティティ(BswModuleEntity)のモードグループ参照 (accessedModeGroup)*は，自身が属する *BSWM ディスクリプション (BswModuleDescription)*に属さなければならない
【nrte_sws_0342】	*BSWM エンティティ（BswModuleEntity）のモードグループ管理 (managedModeGroup)*は，自身が属する *BSWM ディスクリプション (BswModuleDescription)の提供モードグループ(providedModeGroup)*に属さなければならない
【nrte_sws_0343】	*BSW スケジューラブル(BswSchedulableEntity)*の所属*パーティション (EcucPartition)*は，1 つでなければならない
【nrte_sws_0344】	*モード宣言グループ(ModeDeclarationGroup)の初期モード(initialMode)*は，*モード宣言(modeDeclaration)*として保持しているものでなければならない
【nrte_sws_0345】	*モード宣言グループプロトタイプ (ModeDeclarationGroupPrototype)*の所属パーティション*(EcucPartition)*は，1 つでなければならない
【nrte_sws_0346】	*実装データ型 (ImplementationDataType)のカテゴリ (category)* が *TYPE_REFERENCE* の場合には，参照する *実装データ型 (ImplementationDataType)*が定義されていなければならない.
【nrte_sws_0349】	イベントセマンティクスによる S/R 連携では，1 以上のキュー*長(queueLength)*を持つ*イベントセマンティクス受信側連携仕様(QueuedReceiverComSpec)*が定義されていなければならない.（RTEGEN は，0 以下の*イベントセマンティクス受信側連携仕様(QueuedReceiverComSpec)*のキュー*長(queueLength)*を拒否する【rte_sws_2526】を上位仕様とする）
【nrte_sws_0350】	データセマンティクスによる S/R 連携では，*データ要素の初期値(initValue)*と，*無効化ポリシ(InvalidationPolicy)* が *'replace'* の場合の *無効値(InvalidValue)* は，異なっていなければならない.（RTEGEN は，データセマンティクスのデータ*要素の初期値*と，*無効化ポリシ(InvalidationPolicy)* が *'replace'* の場合の *無効値(InvalidValue)*が同じ設定を拒否する【rte_sws_8007】を上位仕様とする）
【nrte_sws_0351】	*ランナブル(RunnableEntity)*と関連付けられるタスクとそのランナブル

仕様番号	制約内容
	*(RunnableEntity)*を保持する SW-C は，同じパーティションに配置されなくてはならない．（RTEGEN は，SW-C の *ランナブル(RunnableEntity)*が異なるパーティションに関連付けられるタスクの設定を拒否する【rte_sws_7347】を上位仕様とする）
【rte_sws_5506】	1:N 連携において，*送信 ACK 要求(TransmissionAcknowledgementRequest)*を定義してはならない．
【rte_sws_7181】	コンフィグ対象 RTE イベント(RteEventRef)が*バックグラウンドイベント(BackgroundEvent)*を参照し，使用 OS アラーム(RteUsedOsAlarmRef)が未定義の場合に，マッピング先 OS タスク(RteMappedToTaskRef)がコア内の最低優先度でない設定を拒否する．
【rte_sws_7403】	ある *SWC 内部振る舞い(SwcInternalBehavior)*において，以下の全ての条件を満たす*データ受信イベント(DataReceivedEvent)*が複数存在してはならない． ・同一の*変数データプロトタイプ(VariableDataPrototype)*を参照しており，*変数データプロトタイプ(VariableDataPrototype)*がイベントセマンティクスである． ・異なるタスクにマッピングされている異なる *ランナブル(RunnableEntity)*を起動する．
【rte_sws_7809】	使用 OS アラーム(RteUsedOsAlarmRef)が定義され，かつ，コンフィグ対象 RTE イベント(RteEventRef)が*周期イベント(TimingEvent)*，または，*バックグラウンドイベント(BackgroundEvent)*を参照していない設定を拒否する．
【nrte_sws_0352】	使用 OS アラーム(RteBswUsedOsAlarmRef)が定義され，かつ，コンフィグ対象 RTE イベント(RteBswEventRef)が*周期イベント(BswTimingEvent)*，または，バックグラウンドイベント(BswBackgroundEvent)を参照していない設定を拒否する
【nrte_sws_0353】	コンフィグ対象 BSW イベント(RteBswEventRef)が*バックグラウンドイベント(BswBackgroundEvent)*を参照し，使用 OS アラーム(RteBswUsedOsAlarmRef)が未定義の場合に，マッピング先 OS タスク(RteBswMappedToTaskRef)がコア内の最低優先度でない設定を拒否する．
【nrte_sws_0354】	マッピング先 OS タスク(RteMappedToTaskRef/RteBswMappedToTaskRef)と使用 OS イベント(RteUsedOsEventRef/RteBswUsedOsEventRef)が一致する全ての RTE イベント-OS タスクマッピング(RteEventToTaskMapping)と BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)について，使用 OS アラーム(RteUsedOsAlarmRef/RteBswUsedOsAlarmRef)が一致していなければならない．
【nrte_sws_0355】	マッピング先 OS タスク(RteMappedToTaskRef/RteBswMappedToTaskRef)と使用 OS イベント(RteUsedOsEventRef/RteBswUsedOsEventRef)が一致する全

仕様番号	制約内容
	ての RTE イベント-OS タスクマッピング(RteEventToTaskMapping)と BSW イベント-OS タスクマッピング(RteBswEventToTaskMapping)について， いずれかの RTE イベント-OS タスクマッピング/BSW イベント-OS タスクマッピングがコンフィグ対象 RTE イベント(RteEventRef/RteBswEventRef)に*バックグラウンドイベント(BackgroundEvent/BswBackgroundEvent)*もしくは*周期イベント(TimingEvent/BswTimingEvent)*を指定している場合，全てのコンフィグ対象 RTE イベントはバックグラウンドイベントもしくは周期イベントでなくてはならない.
【nrte_sws_0356】	コンフィグ対象 RTE イベント(RteEventRef)が*バックグラウンドイベント(BackgroundEvent)*のとき，マッピング先 OS タスク(RteMappedToTaskRef)が定義されていなければならない.
【nrte_sws_0357】	コンフィグ対象 BSW イベント(RteBswEventRef)が*バックグラウンドイベント(BswBackgroundEvent)*のとき，マッピング先 OS タスク(RteBswMappedToTaskRef)が定義されていなければならない.
【nrte_sws_0367】	コンフィグ対象 RTE イベント(RteEventRef)が*データ受信イベント(DataReceivedEvent)*のとき，マッピング先 OS タスク(RteMappedToTaskRef)が定義されていなければならない.
【nrte_sws_0368】	コンフィグ対象 RTE イベント(RteEventRef)が*データ受信エラーイベント(DataReceiveErrorEvent)*のとき，マッピング先 OS タスク(RteMappedToTaskRef)が定義されていなければならない.
【nrte_sws_0369】	コンフィグ対象 RTE イベント(RteEventRef)が*データ送信完了イベント(DataSendCompletedEvent)*のとき，マッピング先 OS タスク(RteMappedToTaskRef)が定義されていなければならない.
【nrte_sws_0370】	マッピング先 OS タスク(RteMappedToTaskRef)と使用 OS イベント(RteUsedOsEventRef)が一致する全ての RTE イベント-OS タスクマッピング(RteEventToTaskMapping)について，いずれかの RTE イベント-OS タスクマッピングがコンフィグ対象 RTE イベント(RteEventRef)に*データ送信完了イベント(DataSendCompletedEvent)*を指定している場合，全てのコンフィグ対象 RTE イベントは，*イベントソース(eventSource)*として同一の*変数データプロトタイプ(VariableDataPrototype)*への参照を持つ*データ送信完了イベント*でなくてはならない.
【nrte_sws_0371】	マッピング先 OS タスク(RteMappedToTaskRef)と使用 OS イベント(RteUsedOsEventRef)が一致する全ての RTE イベント-OS タスクマッピング(RteEventToTaskMapping)について，いずれかの RTE イベント-OS タスクマッピングがコンフィグ対象 RTE イベント(RteEventRef)に*データ受信イベント*

仕様番号	制約内容
	*(DataReceivedEvent)*を指定している場合，全てのコンフィグ対象 RTE イベントは，同一の*変数データプロトタイプ(VariableDataPrototype)*への参照を持つ*データ受信イベント*でなくてはならない.
【nrte_sws_0372】	マッピング先 OS タスク(RteMappedToTaskRef)と使用 OS イベント(RteUsedOsEventRef)が一致する全ての RTE イベント-OS タスクマッピング(RteEventToTaskMapping)について，いずれかの RTE イベント-OS タスクマッピングがコンフィグ対象 RTE イベント(RteEventRef)に*データ受信エラーイベント(DataReceiveErrorEvent)*を指定している場合，全てのコンフィグ対象 RTE イベントは，同一の*変数データプロトタイプ(VariableDataPrototype)*への参照を持つ*データ受信エラーイベント*でなくてはならない.
【nrte_sws_0377】	*パーティション(EcucPartition)*の *BSWM 実行パーティション(EcucPartitionBswModuleExecution)*が真に設定されたパーティションは，信頼パーティションでなくてはならない.
【nrte_sws_0379】	*COM シグナル(ComSignal)*に*ビットサイズ(ComBitSize)*が定義されている場合，*シグナルビットサイズ(ComBitSize)*と，COM シグナルに対応する*実装データ型(ImplementationDataType)*に関連する*ソフトウェアベース型(SwBaseType)*の*ベース型サイズ(baseTypeSize)*は一致しなければならない.
【nrte_sws_0380】	*COM グループシグナル(ComGroupSignal)*に*ビットサイズ(ComBitSize)*が定義されている場合，*シグナルビットサイズ(ComBitSize)*と，COM グループシグナルに対応する*実装データ型(ImplementationDataType)*に関連する*ソフトウェアベース型(SwBaseType)*の*ベース型サイズ(baseTypeSize)*は一致しなければならない.
【nrte_sws_0381】	*COM シグナル(ComSignal)*に*シグナル長(ComSignalLength)*が定義されている，かつ，COM シグナルに対応する*実装データ型(ImplementationDataType)*の*カテゴリ(category)*が ARRAY の場合，*シグナル長(ComSignalLength)*と，*実装データ型(ImplementationDataType)*の*実装データ型要素(ImplementationDataTypeElement)*の*配列サイズ(arraySize)*は一致しなければならない.
【nrte_sws_0382】	*ポート定義引数値(PortDefinedArgumentValue)*の*型(valueType)*は，プリミティブ実装データ型およびその再定義型が定義されていなければならない.(*ポート定義引数値(PortDefinedArgumentValue)*の*データ型(valueType)*は，*カテゴリ(category)*が VALUE もしくは TYPE_REFERENCE であり，TYPE_REFERENCE の場合はその*ターゲットカテゴリ(targetCategory)*が VALUE でなければならない【constr_1150】を上位仕様とする)
【nrte_sws_0383】	OS スケーラビリティクラスが SC3/SC4 の場合，**RTE 排他エリア実現メカニズ**

仕様番号	制約内容
	ム(RteExclusiveAreaImplementation)の**排他エリア実現メカニズム手段(RteExclusiveAreaImplMechanism)**は，非信頼 OSAP に所属する処理単位から全割込みのブロック(ALL_INTERRUPT_BLOCKING)を使用してはならない.
【nrte_sws_0384】	時間パーティショニング機能対応 OS の場合，**RTE 排他エリア実現メカニズム(RteExclusiveAreaImplementation)**の**排他エリア実現メカニズム手段(RteExclusiveAreaImplMechanism)**は，非信頼 OSAP に所属する処理単位から全割込みのブロック(ALL_INTERRUPT_BLOCKING)，OS 割込みのブロック(OS_INTERRUPT_BLOCKING)，OS リソースの獲得(OS_RESOURCE)を使用してはならない.

2.16 RTE/SCHM コード生成方針

2.16.1 RTE/SCHM コード生成フロー

本 RTEGEN を使用して，RTE/SCHM を生成する際のフローを以下に示す.

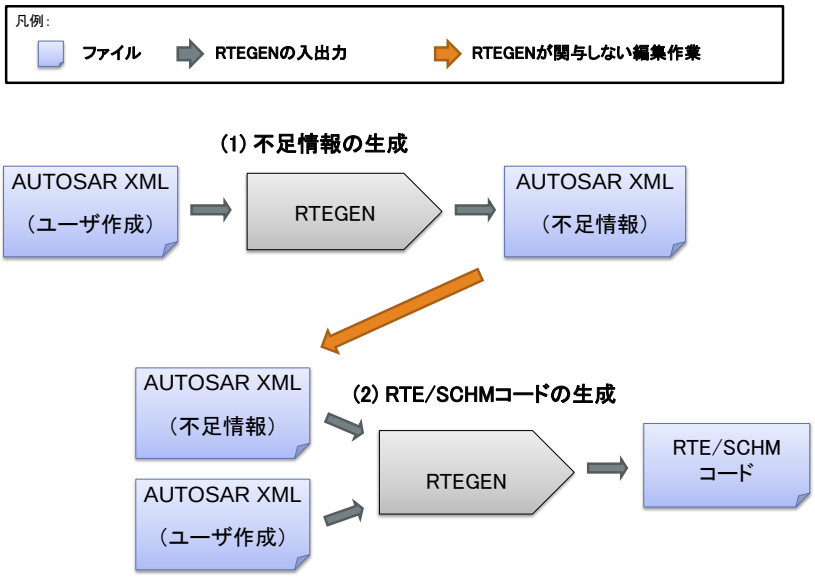

図 2-30 RTE/SCHM を生成する際のフロー

(1) 不足情報の生成

本工程では，入力された AUTOSAR XML に対し，RTE/SCHM を生成する上での不足情報がないかの確認を行い，検出した不足情報を補完する AUTOSAR XML を生成する.

(2) RTE/SCHM コードの生成

本工程では，入力された AUTOSAR XML を元に，RTE/SCHM を生成する．RTE は，ECU インスタンス個別にコンフィグされ，コード生成される【rte_sws_5000】．なお，RTE は，静的な連携のみをサポートする【rte_sws_6026】.

本 RTEGEN は，入力された AUTOSAR XML の情報を元に，工程(1)，もしくは工程(2)のいずれの工程を実施するかを自動的に判断する.

RTE 生成の挙動

本 RTEGEN は，ユーザの入力した AUTOSAR XML に RTE/SCHM を生成する上での不足情報が存在する場合，不足情報を補完する AUTOSAR XML を出力し，RTE/SCHM を生成しない【nrte_sws_0173】．例として，RTEGEN は，IOC モジュールが使用される場合に，IOC モジュールの情報を提供する【rte_sws_5147】ことで，IOC コンフィグ情報で使用される内部実装データ型を生成する【rte_sws_8400】. 不足情報を補完する AUTOSAR XML は，ユーザの入力した AUTOSAR XML からの差分情報のみを含む【nrte_sws_0177】．RTEGEN が決定できない不足情報は，不足情報を補完する AUTOSAR XML に含めない【nrte_sws_0174】．出力された AUTOSAR XML を編集し，RTEGEN により出力されない不足情報を追加するのは，ユーザの責任である.

本 RTEGEN は，ユーザの入力した AUTOSAR XML に RTE/SCHM を生成する上での不足情報が存在しない場合，RTE/SCHM コードを生成する【nrte_sws_0175】．生成される RTE/SCHM API とコードは，入力情報が同じである限り同じである．【rte_sws_2514】.

RTE/SCHM コードを生成する上での不足情報は，ベンダ毎の RTE/SCHM の実装に依存するため，実装定義とする【nrte_sws_0176】.

2.16.2 RTE/SCHM コード生成条件

本 RTEGEN は，入力された AUTOSAR XML に含まれるコンフィグ情報の内容に応じて，RTE/SCHM コード生成要否を判別する【nrte_sws_0223】(表 2-40).

AUTOSAR 仕様との違い

AUTOSAR 仕様では，SCHM のみを生成するフェーズがある【rte_sws_7569】．このフェーズでは，RTE コンフィグ情報が入力された場合，RTEGEN はその入力を違反にすると規定されている【rte_sws_7585】しかし，A-RTEGEN では，SCHM のみを生成するフェーズはサポートしない【irte_sws_0011】.

表 2-40 RTE/SCHM コード生成条件

RTEGEN の入力		RTEGEN の出力
RTE コンフィグ情報 存在有無	SCHM コンフィグ情報 存在有無	
あり	あり	RTE/SCHM コード(図 2-31)
あり	なし	RTE コード(図 2-32)
なし	あり	SCHM コード(図 2-33)
なし	なし	コード生成しない． コンフィギュレーション違反 〔nrte_sws_0111〕〔nrte_sws_0226〕.

ここで，RTE/SCHM コンフィグ情報の存在判定は以下の通りである.

表 2-41 RTE/SCHM コンフィグ情報の存在判定

モジュール	コンフィグ情報存在有無	存在判定条件	
		判定対象コンフィグ情報	有無
RTE	あり	*ルートソフトウェアコンポジション (rootSoftwareComposition)*	あり
	なし		なし
SCHM	あり	*BSW 実装(BswImplementation)*	あり
	なし		なし

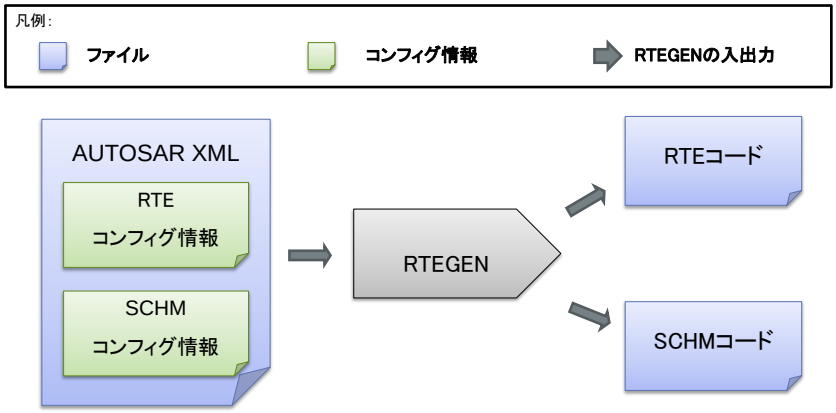

図 2-31 RTE/SCHM コード生成条件

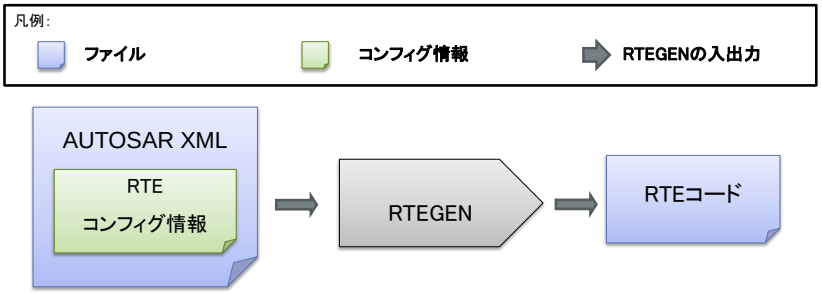

図 2-32 RTE コード生成条件

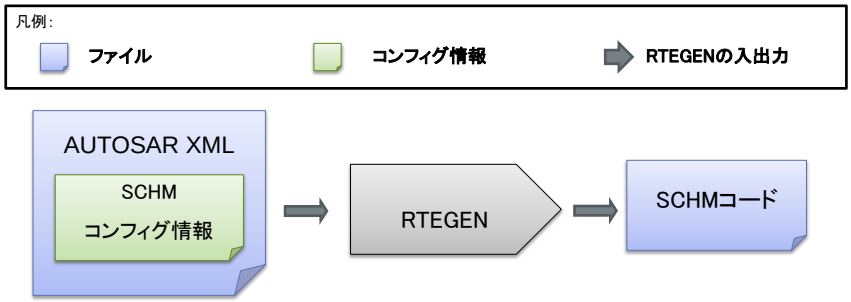

図 2-33 SCHM コード生成条件

2.16.3 処理モード

RTEGEN の処理モードには，以下の 2 つがある．

処理モード	説明
互換モード	RTEGEN のデフォルトの処理モードで，明確に定義された「標準化」データ構造体の使用を通じて異なるベンダ間での RTEGEN の互換性を保証する． 互換モードでコンパイルした SW-C は，異なるベンダの RTE 上で動作させることができる．
ベンダモード	ベンダモードは異なるベンダ間での RTE の互換性をなくして，固有実装つまり効率のいいデータ構造や型を許可する． ベンダモードでコンパイルした SW-C は，異なるベンダの RTE 上で動作せず，同じベンダの RTE 上でのみ動作する．

SW-C と RTE のソースコード互換性

いずれの処理モードにおいても，生成された RTE は，SW-C のソースコードと互換していなければならない【rte_sws_1195】．RTE は，SW-C のソースコードが提供されている限り，どの処理モードでもその SW-C が使用できることを保証する．

処理モードの設定

処理モードは，**RTE 生成パラメータ(RteGeneration)**の **RTE 生成モード(RteGenerationMode)**により指定する【rte_sws_a_0048】．A-RTEGEN では，**RTE 生成モード(RteGenerationMode)**は常に COMPATIBILITY_MODE に設定されているものとして扱い，**RTE 生成モード(RteGenerationMode)**のコンフィギュレーションはサポートしない【irte_sws_0003】．

2.16.3.1 互換モード

互換モードはデフォルトの処理モードであり，全ての RTEGEN が互換モードをサポートする【rte_sws_1151】．

RTEGEN は，以下の互換性を保証するアプリケーションヘッダ，および RTE を生成する【rte_sws_1216】．

互換モードで生成されたコントラクトフェーズのアプリケーションヘッダでコンパイルされた SW-C は，互換モードで生成された RTE とリンクして動作する．

2.16.3.2 ベンダモード

RTEGEN はベンダモードをサポートしなくともよい【rte_sws_1152】．本 RTEGEN では，ベンダモードをサポートするが，ベンダモードで生成された RTE でどのような最適化が行われるかは実装定義とする【nrte_sws_0044】．

要求 RTE ベンダの設定

要求 RTE ベンダは，*SW-C 実装(SwcImplementation)*の*要求 RTE ベンダ(requiredRTEVendor)*により指定する【nrte_sws_0046】．A-RTEGEN では，*要求 RTE ベンダ(requiredRTEVendor)*は常に本 RTE のベンダ ID が設定されているものとして扱い，*要求 RTE ベンダ(requiredRTEVendor)*のコンフィギュレーションはサポートしない【irte_sws_0005】．

要求 RTE ベンダの指定がない場合，RTEGEN は，SW-C が互換モードでコンパイルされたものとして扱う【rte_sws_1234】．

要求 RTE ベンダの指定がある場合，RTEGEN は，SW-C がベンダモードでコンパイルされたものとして扱う【nrte_sws_0045】．

2.16.4 最適化モード

生成される RTE の最適化を行う方針として，以下の 2 つの方針をサポートする．RTEGEN は，最適化モードの方針に従い最適化したコードを生成する【rte_sws_5053】．どちらの方針を使用するかは，コンフィギュレーションにより選択できる．

最適化モード	説明
消費メモリ優先(MEMORY)	消費メモリが最小になるよう最適化を行う．
実行時間優先(RUNTIME)	実行時間が最小になるよう最適化を行う．

最適化モードの設定

最適化モードは，**RTE 生成パラメータ(RteGeneration)**の **RTE 最適化モード(RteOptimizationMode)**により指定する【rte_sws_a_0049】．A-RTEGEN では，**RTE 最適化モード(RteOptimizationMode)**は常に実行時間優先(RUNTIME)に設定されているものとして扱い，**RTE 最適化モード(RteOptimizationMode)**のコンフィギュレーションはサポートしない【irte_sws_0004】．

2.16.5 ビルドサポート

RTE のソースコードは，ビルド時に SW-C，および BSWM とリンクするために本節の規則を守る必要がある．

RTE/SCHM 内部メモリに対するメモリマッピング

RTE/SCHM が割り当てるメモリオブジェクト(関数，およびグローバル変数)は，"Rte"を接頭辞とするメモリマッピングのマクロでラップして定義する【rte_sws_5088】．メモリマッピングについては，関連文書 Specification of Memory Mapping を参照．

AUTOSAR データプロトタイプ実装のメモリオブジェクトのメモリマッピング

*AUTOSAR データプロトタイプ(AutosarDataPrototype)*の実装のためのメモリオブジェクトに対するメモリマッピングの定義は以下のルールに従う．

- 〔rte_sws_7592〕の場合を除き，メモリ確保キーワードのは，*ソフトウェアアドレッシング方式(SwAddrMethod)*のショートネームとなる【rte_sws_7589】.
- *ソフトウェアアドレッシング方式(SwAddrMethod)の定義されていない AUTOSAR データプロトタイプ(AutosarDataPrototype)の実装のためにメモリオブジェクトが生成される場合，*メモリ確保キーワードのは，RTEGEN が任意に決めてよい【rte_sws_a_0051】.
- *ソフトウェアアドレッシング方式(SwAddrMethod)のメモリ確保キーワードポリシ(memoryAllocationKeywordPolicy)*が AddrMethodShortName である場合，メモリ確保キーワードの<ALIGNMENT>接尾辞は省略する【rte_sws_7047】.
- *ソフトウェアアドレッシング方式(SwAddrMethod)のメモリ確保キーワードポリシ(memoryAllocationKeywordPolicy)*が AddrMethodShortNameAndAlignment である場合，メモリ確保キーワードの<ALIGNMENT>接尾辞は〔rte_sws_7049〕，〔rte_sws_7050〕，〔rte_sws_7052〕，および〔rte_sws_7053〕で規定されるアラインメントとなる【rte_sws_7048】.
- *AUTOSAR データプロトタイプ(AutosarDataPrototype)のソフトウェアアラインメント(swAlignment)*は，〔rte_sws_7050〕，〔rte_sws_7052〕，および〔rte_sws_7053〕で規定される*実装データ型(ImplementationDataType)*のアラインメントに優先して使用する【rte_sws_7049】.
- プリミティブ実装データ型または配列実装データ型である *AUTOSAR データプロトタイプ(AutosarDataPrototype)*のアライメントは，*ソフトウェアベース型(SwBaseType)のベース型サイズ(baseTypeSize)*となる【rte_sws_7050】.
- 再定義実装データ型である *AUTOSAR データプロトタイプ(AutosarDataPrototype)*のアライメントは，再定義された実装データ型のアライメントと同じとなる【rte_sws_7052】.
- ポインタ実装データ型である *AUTOSAR データプロトタイプ(AutosarDataPrototype)*のアライメントは不定とする【rte_sws_7053】.
- RTEGEN が *AUTOSAR データプロトタイプ(AutosarDataPrototype)*の実装のために複数のメモリオブジェクトを生成する場合，追加のメモリオブジェクトのは RTEGEN が任意に決めてよい【rte_sws_7592】.
- 提供側ポートの *AUTOSAR データプロトタイプ(AutosarDataPrototype)のソフトウェアアドレッシング方式(SwAddrMethod)*は，要求側ポートの *AUTOSAR データプロトタイプ(AutosarDataPrototype)のソフトウェアアドレッシング方式(SwAddrMethod)*に優先して使用する【rte_sws_7590】.
- 以下の全ての条件を満たす場合，本 RTEGEN では，メモリ確保キーワードのは〔nrte_sws_0180〕に準拠し，〔nrte_sws_0180〕におけるグローバル変数の所属パーティションは要求側ポートの *AUTOSAR データプロトタイプ(AutosarDataPrototype)*の所属するパーティション(EcucPartition)のショートネームとする【nrte_sws_0162】.
  - 〔rte_sws_7592〕〔rte_sws_a_0051〕のいずれかを満たす.
  - パーティション構成である.
  - メモリオブジェクトがグローバル変数である.

- 以下の全ての条件を満たす場合，本 RTE では，メモリ確保キーワードのは〔nrte_sws_0181〕に準拠する【nrte_sws_0163】.
  - 〔rte_sws_7592〕〔rte_sws_a_0051〕のいずれかを満たす.
  - 非パーティション構成である.
  - メモリオブジェクトがグローバル変数である.
- 以下の全ての条件を満たす場合，本 RTEGEN では，メモリ確保キーワードのは〔nrte_sws_0164〕に準拠し，〔nrte_sws_0164〕における関数の所属パーティションは*AUTOSAR データプロトタイプ(AutosarDataPrototype)*の所属する*パーティション(EcucPartition)*とする【nrte_sws_0178】.
  - 〔rte_sws_7592〕を満たす.
  - パーティション構成である.
  - 追加のメモリオブジェクトが関数である.
- 以下の全ての条件を満たす場合，本 RTE では，メモリ確保キーワードのは〔nrte_sws_0165〕に準拠するものとする【nrte_sws_0179】.
  - 〔rte_sws_7592〕を満たす.
  - 非パーティション構成である.
  - 追加のメモリオブジェクトが関数である.

RTEGEN が生成するその他のメモリオブジェクトのメモリマッピング

RTEGEN が生成するその他のグローバル変数のメモリマッピングの定義は以下のルールに従う.

- パーティション構成の場合，本 RTEGEN では，メモリ確保キーワードのは"VAR_<PID>_INIT"とする【nrte_sws_0180】．ここで，<PID>は，グローバル変数の所属するパーティション(EcucPartition)のショートネーム.
- 非パーティション構成の場合，本 RTEGEN では，メモリ確保キーワードのは"VAR_INIT"とする【nrte_sws_0181】.

RTEGEN が生成するその他の関数のメモリマッピングの定義は以下のルールに従う.

- パーティション構成の場合，本 RTEGEN では，メモリ確保キーワードのは"CODE_<PID>"とする【nrte_sws_0164】．ここで，<PID>は，関数の所属するパーティション(EcucPartition)のショートネーム.
- 非パーティション構成の場合，本 RTEGEN では，メモリ確保キーワードのは"CODE"とする【nrte_sws_0165】.

3. API 仕様

3.1　API 方針

3.1.1　基本方針

RTE/SCHM のコンフィギュレーション／生成単位

RTE/SCHM は，ECU 毎にコンフィギュレーション，および生成される【rte_sws_1316】．ECU 毎に最適なコンフィギュレーションを行うことにより，RTE/SCHM の実行効率の向上とメモリ消費の軽減を可能にする．

RTE/SCHM API の基本方針

RTE/SCHM API の設計は以下の方針に従う．

- API がある処理を達成する唯一の方法であること(例えば，あるデータ要素の送信のためには，単一の API を提供する)．
- API はコンパイラに依存しないこと【rte_sws_1314】．
- RTE/SCHM 実装は AUTOSAR の提供するコンパイラ抽象化メカニズムを使用すること【rte_sws_3787】．
- API はソースコード，もしくはオブジェクトコードのみが提供された SW-C をサポートすること【rte_sws_1315】．

3.1.2　プログラム言語

RTE/SCHM は，C/C++で記述された SW-C，および BSWM をサポートする．

RTE/SCHM の実装に使用する言語

RTE/SCHM は C 言語で記述する【rte_sws_1167】．

RTE/SCHM コードの MISRA 適合性

全ての RTE/SCHM コードは，以下の条件で MISRA C 規格に従う必要がある【rte_sws_1168】．

- RTE/SCHM コードは，MISRA C 標準の HIS サブセットに適合していなければならない．
- 技術的に正当な理由がある場合，例外的に MISRA 違反ケースが認められる．
- MISRA 規則#11 を除き，違反ケースは明確に識別され，文書化される必要がある．

参考：MISRA 規則#11

識別子(内部・外部)は 31 文字を超えて識別できることを想定してはならない．さらにコンパイラ・リンカが外部識別子に対して 31 文字までの識別および大文字小文字の区別をサポートすることを保証しているかチェックしなければならない．

C++で記述された SW-C のサポート

3.7 節の API は C++で記述された SW-C からも直接アクセスできる．

RTE は，C++で記述された SW-C に，全ての RTE シンボルが C 言語リンケージとしてインポートされることを保証する【rte_sws_1011】．RTE API のシンボルのインポートはアプリケーションヘッダで行われる．

3.1.3　RTE 名前空間

接頭辞"Rte_"を使用したグローバル名前空間内のシンボル名は RTE のために予約されており，RTE はこの接頭辞をシンボルの重複を避けるために使用する．

RTE の外部シンボルは，接頭辞"Rte_"を使用する．この規則は，以下のシンボルには適用しない【rte_sws_1171】．

- 以下の仕様で規定される AUTOSAR データ型で表現される型名．
  - プリミティブ実装データ型〔rte_sws_7104〕
  - 配列実装データ型〔rte_sws_7110〕
  - 構造体実装データ型〔rte_sws_7114〕
  - 共用体実装データ型〔rte_sws_7144〕
  - 再定義実装データ型〔rte_sws_7109〕
  - ポインタ実装データ型〔rte_sws_7148〕
- 列挙定数〔rte_sws_3810〕．
- 上限値，および下限値定数〔rte_sws_5052〕．

3.1.4　SCHM 名前空間

SCHM の名前空間は，以下を除き，RTE の名前空間を使用する．

- SCHM の外部シンボル(例. 関数名，データ型等)は，接頭辞"SchM_"を使用する【rte_sws_7284】.

3.1.5　API マッピング

RTE は，RTE API 名から RTE API の実体として生成される関数(生成関数)へのマッピング(API マッピング)を提供する． ECU の実行バイナリ内で生成関数名はユニークでなければならないため，API マッピングは必要である．RTEGEN は，API マッピングをアプリケーションヘッダに生成する【rte_sws_1274】．

3.1.5.1　コントラクトフェーズ

コントラクトフェーズでは，API マッピングは API 呼出しを，RTEGEN が生成した API 実装に変換する必要がある．コントラクトフェーズの API マッピングは，RTE/SCHM API 呼出しの生成関数へのマッピングを提供する【rte_sws_3707】．

<u>マッピング先の生成関数の命名則</u>

Rte_<name>_<api_extension>形式の RTE API に対応する生成関数名は以下の規則に従う【rte_sws_3837】．

Rte_<name>_<c>_<api_extension>

ここで,

- <name>は API ルート名(例えば，Rte_Write であれば，Write),
- <c>は SW-C 型のショートネーム,
- <api_extension>は<name>に依存する API の拡張子(例えば，Rte_Write であれば，<p>_<o>)を表す.

API マッピングの生成例

SW-C<c1>のクライアントポート<p1>から，引数 1 つのオペレーション<a>を行う Rte_Call の場合，以下の API マッピングが生成される.

#define Rte_Call_p1_a Rte_Call_c1_p1_a

3.1.5.2　ジェネレーションフェーズ

コントラクトフェーズで生成された API マッピングのマッピング先である生成関数は，ジェネレーションフェーズにおいて生成する.

前述したように，API マッピングは API 名と適切な生成関数をマッピングする責任がある. RTEGEN は生成関数を提供する代わりに，より効率的な実装関数を提供してもよい【rte_sws_a_0044】. この場合，API マッピングは，RTE/SCHM API から RTE/SCHM 内の最適化された実装関数への中継関数として動作する．本 RTEGEN では，効率的な実装関数の提供による最適化を行わない【nrte_sws_0041】.

マッピング先の生成関数の引数

生成関数は，API マッピングと同じ順序で同じ引数を取る【rte_sws_1153】.

3.1.5.3　関数の省略

ジェネレーションフェーズの API マッピングを使用すると，関数を使用せず，マクロまたはインライン関数での RTE/SCHM API を実装することが可能になる.

API マッピングは関数を使用せず，マクロのみの RTE/SCHM API 実装を提供してもよい【rte_sws_a_0045】．本 RTEGEN では，マクロのみの RTE/SCHM API 実装を提供しない【nrte_sws_0042】.

API マッピングが RTE/SCHM API のための生成関数を使用しない場合，ジェネレーションフェーズの API マッピングは，RTE/SCHM API の呼出し元 SW-C/BSWM に対し返り値を返し，SW-C/BSWM に変更なく RTE/SCHM API が使用できることを保証する【rte_sws_1146】.

*ポートプロトタイプ(PortPrototype)のポート API オプション(PortAPIOption)のアドレス取得有効(enableTakeAddress)*が true の場合，*そのポートプロトタイプ(PortPrototype)*に関連する RTE API は関数で提供されなければならない【rte_sws_7100】.

本 RTEGEN は，インライン化することで高速化の効果が高い以下の RTE API に対して，インライ

ン関数実装を提供する【nrte_sws_0273】.

<table>
<tr><th>RTE API</th><th colspan="2">条件 1</th><th colspan="2">条件 2</th><th>インライン化有無</th></tr>
<tr><td>Rte_Write</td><td rowspan="4">実現方式</td><td rowspan="3">RTE 内部バッファの参照/更新のみで高速化の効果が高い(※)</td><td rowspan="3">ポート API オプション(PortAPIOption)のアドレス取得有効(enableTakeAddress)</td><td>true</td><td>×</td></tr>
<tr><td>Rte_Read</td><td rowspan="2">false</td><td rowspan="2">○</td></tr>
<tr><td>Rte_Invalidate</td></tr>
<tr><td></td><td>上記以外</td><td colspan="2">－</td><td>×</td></tr>
<tr><td>Rte_Send</td><td colspan="2">－</td><td colspan="2">－</td><td>×</td></tr>
<tr><td>Rte_Receive</td><td colspan="2">－</td><td colspan="2">－</td><td>×</td></tr>
<tr><td rowspan="3">Rte_Call</td><td rowspan="3">SW-C 間の連携パターン</td><td>パーティション内連携</td><td rowspan="3">ポート API オプション(PortAPIOption)のアドレス取得有効(enableTakeAddress)</td><td>true</td><td>×</td></tr>
<tr><td>パーティション間連携(コア内)，かつ，クライアント(信頼パーティション)／サーバ(信頼パーティション)</td><td>false</td><td>○</td></tr>
<tr><td>上記以外</td><td colspan="2">－</td><td>×</td></tr>
<tr><td>Rte_IrvRead</td><td colspan="2">－</td><td colspan="2">－</td><td>○</td></tr>
<tr><td>Rte_IrvWrite</td><td colspan="2">－</td><td colspan="2">－</td><td>○</td></tr>
<tr><td>Rte_Enter</td><td colspan="2">－</td><td colspan="2">－</td><td>○</td></tr>
<tr><td>Rte_Exit</td><td colspan="2">－</td><td colspan="2">－</td><td>○</td></tr>
</table>

(※) 高速化の効果が高い実現方式の具体的な選択条件は実装定義とする【nrte_sws_0274】.

関数を使用しない API マッピングの生成例

ポート p1 のデータ要素 a に対する Rte_Send の場合，ジェネレーションフェーズの API マッピングは以下のように記述できる.

#define Rte_Send_p1_a(s, a)　(<var> = (a), RTE_E_OK)

ここで，<var>は，送信側-受信側間の連携のために，RTE が生成するバッファのシンボル名である.

3.1.5.4　API 命名規則

SW-C はポート経由でその他の SW-C，および BSWM と連携を行うため，RTE API 名は以下を組み合わせた名前となる【rte_sws_a_0051】.

・ API のルート名(Rte_Write であれば，Write).
・ API が動作する箇所(以下，API アクセスポイント).

ポート経由で動作する API の場合，API アクセスポイントはポート名を含む.

S/R インタフェースは複数のデータ要素，C/S インタフェースは複数のオペレーションをサポートする．そのため RTE API は，API アクセスポイント上のデータ要素名やオペレーション名を含むことによって，ポート上のどのデータ要素，もしくはオペレーションにアクセスするかを示す.

また RTE API は，RTE の名前空間内に含まれるため，シンボル名は接頭辞"Rte_"で始まる.

3.1.5.5　API 引数

API 引数は以下のいずれかに分類される.

API 引数の分類	内容
IN 引数	API への入力引数．API 関数により内容が参照される.
OUT 引数	API の処理結果の出力引数．API 関数により内容が更新される.
INOUT 引数	API への入力，および処理結果の出力の両方に使用する引数．API 関数により内容が参照，および更新される.

"In"引数の値の引き渡し方法

IN 引数の値の引き渡し方法は以下の通りである．const 修飾子のセマンティクスは P2CONST マクロの仕様に従う.

実装データ型	値の引き渡し方法	const 修飾子付与有無
プリミティブ実装データ型	値渡し【rte_sws_1017】	×
構造体実装データ型	参照渡し【rte_sws_1018】	○【rte_sws_7086】
共用体実装データ型	参照渡し〔rte_sws_1018〕	○〔rte_sws_7086〕
配列実装データ型	参照渡し(配列表現)【rte_sws_5107】	○〔rte_sws_7086〕
ポインタ実装データ型	値渡し【rte_sws_7661】	×
再定義実装データ型	再定義実装データ型が参照する実装データ型の型に従う【rte_sws_7084】	

"Out"引数への参照先の引き渡し方法

OUT 引数の値の引き渡し方法は以下の通りである.

実装データ型	値の引き渡し方法
プリミティブ実装データ型	参照渡し【rte_sws_1019】
構造体実装データ型	参照渡し【rte_sws_7082】
共用体実装データ型	参照渡し〔rte_sws_7082〕
配列実装データ型	参照渡し(配列表現)【rte_sws_5108】
ポインタ実装データ型	参照渡し【rte_sws_7083】
再定義実装データ型	再定義実装データ型が参照する実装データ型の型に従う〔rte_sws_7084〕

"In/Out"引数への参照先の引き渡し方法

INOUT 引数の値の引き渡し方法は以下の通りである.

実装データ型	値の引き渡し方法
プリミティブ実装データ型	参照渡し【rte_sws_1020】
構造体実装データ型	参照渡し〔rte_sws_1020〕
共用体実装データ型	参照渡し〔rte_sws_1020〕
配列実装データ型	参照渡し(配列表現)【rte_sws_5109】
再定義実装データ型	再定義実装データ型が参照する実装データ型の型に従う〔rte_sws_7084〕

3.1.5.6　API 返り値

API のシグネチャに返り値(<return>)がある場合，RTEGEN は*データプロトタイプ(DataPrototype)*の*実装データ型(ImplementationDataType)*に従って，返り値の型を決めなければならない【rte_sws_7069】.

返り値の型と返す値は以下の通りである.

実装データ型	返り値の型	返す値
プリミティブ実装データ型	*実装データ型(ImplementationDataType)*のショートネーム【rte_sws_7070】	*データプロトタイプ(DataPrototype)*の値〔rte_sws_7070〕
再定義実装データ型	再定義実装データ型が参照する実装データ型の型に従う【rte_sws_7074】	

3.1.5.7　エラー処理

RTE は，以下の 2 つのエラーを扱う．

エラー分類	内容
通信基盤エラー	資源獲得の失敗，もしくは不当な入力パラメータによって発生するエラー．このエラーは通信の失敗時等に BSWM，もしくはハードウェアで発生する．
アプリケーションエラー	SW-C によって通知されるエラー．

通信基盤エラーとアプリケーションエラーのどちらであるかは，SW-C から識別可能である．

RTE は以下の契機で発生したアプリケーションエラーを扱うことができる．

- C/S 連携におけるサーバランナブルの返り値．
- データセマンティクスの S/R 連携におけるデータ無効化．

RTE では，エラーおよび状態に関する情報は Std_ReturnType 型により指定する．

RTE API の呼び出しにおいて検出したエラーは API の返り値で呼び出し側に通知する．RTE は，エラー状態(エラーなし状態を含む)を，RTE API の返り値としてのみ SW-C に渡す【rte_sws_1034】．

ある RTE API の呼び出しは通信サービスで非同期に動作する場合がある．その場合，API からの返り値は API 呼出し中に検出したエラーのみを示す．API 呼び出し完了後に検出したエラーは別の API により通知する．

3.1.6　同一でないポートインタフェース間の接続の扱い

提供側インタフェースのデータ要素やオペレーションのうち，要求側インタフェースに存在しないような要素を，「未接続要素」と定義する．

提供側インタフェースのデータ要素，もしくはオペレーションに対して，「未接続要素」となるような要求側インタフェースのデータ要素，もしくはオペレーションを，接続されていないものとして扱う【rte_sws_1369】．

以下の条件を満たす提供側データ要素について，Rte_Send，および Rte_Write はポートが未接続であるように動作する【rte_sws_1370】．

- S/R 連携の送信側ポートのあるデータ要素について，全ての接続先の要求側ポートにおいて「未接続要素」となる．

3.2　API 仕様記載凡例

API 仕様記載の凡例を示す.

API 名

C 言語 I/F	＜C 言語 I/F を記載する＞
パラメータ[in]	＜API パラメータで入力となるものを記載する＞
パラメータ[in/out]	＜API パラメータで入力と出力となるものを記載する＞
パラメータ[out]	＜API パラメータで出力となるものを記載する＞
返り値	＜API の返り値で標準エラーとなるものと，該当エラーが発生する条件を記載する．(W)と付いている場合は警告として扱ってもよい＞
生成有無の設定	＜API を生成するための設定方法を記載する＞
機能	＜API の機能を記載する＞

データ型名

データ型名	＜データ型の ID 名を記載する＞
概要	＜データ型が示す内容を記載する＞

マクロ名

マクロ名	＜マクロの ID 名を記載する＞
概要	＜マクロの定義内容を記載する＞

定数名

定数名	＜定数の ID 名を記載する＞
概要	＜定数が示す内容を記載する＞

コンテナ名＜コンテナ名称の末尾のみを記載する＞

コンテナ名	＜コンテナ名称をフルパスで記載する＞
概要	＜コンテナの概要を記載する＞
多重度	＜コンテナの多重度を記載する＞
パラメータ	＜コンテナが含むパラメータ名を列挙する＞
	：
サブコンテナ	＜コンテナが含むサブコンテナを列挙する＞
	：

コンテナパラメータ名＜コンテナパラメータの末尾のみを記載する＞

パラメータ名	＜コンテナパラメータ名称をフルパスで記載する＞
概要	＜コンテナパラメータの概要を記載する＞
型	＜コンテナパラメータのデータ型を記載する＞
値の範囲	＜コンテナパラメータの値範囲を記載する＞
多重度	＜コンテナパラメータの多重度を記載する＞
制限事項	＜制限事項がある場合，コンテナパラメータへの制約を記載する＞

3.3　API データ型

3.3.1　Std_ReturnType

データ型名	Std_ReturnType【rte_sws_a_0046】
概要	API 関数が返す「状態」と「エラー値」

Std_ReturnType の構造

この型は uint8 で定義されており，値"0"はエラーが発生しなかったことを示すために使用する．

上位	7	即時通信基盤エラーフラグ 1：即時通信基盤エラーのエラー値である． 0：即時通信基盤エラーのエラー値でない． (つまりアプリケーションエラー)
	6	オーバーレイエラーフラグ (このフラグの使用方法は返り値を返す API に依存する)
	5	アプリケーションエラーの値に使用する
	4	
	3	
	2	
	1	
下位	0	

3.3.1.1　通信基盤エラー

通信基盤エラーは以下の 2 種類に分類される．

エラー分類	内容
即時通信基盤エラー	現在使用しているデータに関連するエラーを示す．このエラーは，アプリケーションデータやアプリケーションエラーが受信できないようなエラーに対して，受信側で使用するエラーである． Std_ReturnType 型の即時通信基盤エラーフラグがセットされている場合，このエラーであることを示す． このエラーは排他的であり，即時通信基盤エラーはアプリケーションエラーを上書きする【rte_sws_2593】．また同時に複数の即時通信基盤エラーが返されることはない．
オーバーレイエラー	現在使用しているデータの受信後に発生した通信イベント(例えばデータ要素が古い，あるいはキューのオーバーフローによるデータ損失)に関連するエラーを示す． オーバーレイエラーは Std_ReturnType 型に含まれるオーバーレイエラーフラグビットを使用して通知する【rte_sws_1318】． このエラーは他のアプリケーションエラー，もしくは通信基盤エラーと併用できる．

3.3.1.2　アプリケーションエラー

RTE は，以下のフォーマットで定義されたアプリケーションエラーをサポートする【rte_sws_2573】

・　即時通信基盤エラーフラグが"0"に設定された Std_ReturnType の下位 6 ビット．

アプリケーションエラーの値範囲

アプリケーションエラーは以下の範囲をとる．

最小値	1
最大値	63

アプリケーションエラーのエラーコード

アプリケーションエラーのエラーコードは，RTE_E_INVALID を除きアプリケーションエラー定数で定義される．アプリケーションエラー定数については，3.5.2 節を参照．

C/S 連携の返り値

C/S 連携では，サーバランナブルはアプリケーションエラー値の範囲内でエラーを返すことができる．

クライアントは以下のどれか 1 つを受信する．

・　通信が成功しなかったことを示す即時通信基盤エラー

・ サーバランナブルの返した返り値
・ サーバランナブルの返した返り値＋オーバーレイフラグを設定した値.

RTE の現在のリリースでは，クライアントに「サーバランナブルの返した返り値＋オーバーレイフラグを設定した値」を返すことはない.

3.3.1.3 定義済みエラーコード

RTE 標準エラーコード定義

RTE API が正常に終了した場合の Std_ReturnType の値は RTE_E_OK(=0)である【rte_sws_1058】.
それ以外の場合はエラーコードで，以下の値を定義する.

・ RTE_E_INVALID = 1【rte_sws_2594】
・ RTE_E_COM_STOPPED = 128【rte_sws_1060】
・ RTE_E_TIMEOUT = 129【rte_sws_1064】
・ RTE_E_LIMIT = 130【rte_sws_1317】
・ RTE_E_NO_DATA = 131【rte_sws_1061】
・ RTE_E_TRANSMIT_ACK = 132【rte_sws_1065】
・ RTE_E_NEVER_RECEIVED = 133【rte_sws_7384】
・ RTE_E_UNCONNECTED = 134【rte_sws_7655】
・ RTE_E_IN_EXCLUSIVE_AREA = 135【rte_sws_2739】
・ RTE_E_SEG_FAULT = 136【rte_sws_2757】
・ RTE_E_DEV_DEFECT = 191【nrte_sws_0270】
・ RTE_E_LOST_DATA = 64【rte_sws_2571】
・ RTE_E_MAX_AGE_EXCEEDED = 64【rte_sws_2702】

以上のエラーコード定義で規定された標準エラーは，RTE ヘッダで定義される【rte_sws_1269】.

SCHM 標準エラーコード定義

SCHM API が正常に終了した場合の Std_ReturnType の値は SCHM_E_OK(=0)である【rte_sws_7289】.
それ以外の場合はエラーコードで，以下の値を定義する.

・ SCHM_E_LIMIT = 130【rte_sws_7290】
・ SCHM_E_NO_DATA = 131【rte_sws_7562】
・ SCHM_E_TRANSMIT_ACK = 132【rte_sws_7563】
・ SCHM_E_IN_EXCLUSIVE_AREA = 135【rte_sws_2747】
・ SCHM_E_TIMEOUT = 129【rte_sws_7054】
・ SCHM_E_DEV_DEFECT = 191【nrte_sws_0335】

以上のエラーコード定義で規定された標準エラーは，RTE ヘッダで定義される【rte_sws_7291】.

3.3.2　プリミティブ実装データ型

データ型名	<name>【rte_sws_7104】
概要	RTE API で使用するデータ型
備考	<name>は，プリミティブ実装データ型のショートネーム

プリミティブ実装データ型を提供するヘッダ

プリミティブ実装データ型は，RTE タイプヘッダ内に定義する〔rte_sws_7104〕．

プリミティブ実装データ型の設定

プリミティブ実装データ型は，*カテゴリ(category)*が VALUE の*実装データ型(ImplementationDataType)*として定義する．

プリミティブ実装データ型の*ベース型(BaseType)*が*ネイティブ宣言(nativeDeclaration)*を持つ場合，RTE タイプヘッダは以下の型宣言を含む〔rte_sws_7104〕．

```
typedef <nativeDeclaration> <name>;
```

ここで，

- <nativeDeclaration>は，*ベース型(BaseType)のネイティブ宣言(nativeDeclaration)*を表す．
- <name>は，プリミティブ実装データ型のショートネームを表す．

本 RTE は，プリミティブ実装データ型の*ベース型(baseType)のベース型エンコーディング(baseTypeEncoding)*に NONE，もしくは BOOLEAN が指定された場合，プリミティブ実装データ型が符号なしであると見なす．NONE，および BOOLEAN 以外が指定された場合，プリミティブ実装データ型が符号ありであると見なす【nrte_sws_0158】．

同一のデータ型の再定義の防止

ショートネーム，および*ベース型(BaseType)のネイティブ宣言(nativeDeclaration)*が等しい複数のプリミティブ実装データ型が定義されている場合，RTE タイプヘッダは〔rte_sws_7104〕に従った型宣言を 1 度だけ含む【rte_sws_7105】．

定義済みのデータ型の再定義防止

RTEGEN は，*ネイティブ宣言(nativeDeclaration)*を持たない*ベース型(BaseType)*に対する型宣言を生成しない【nrte_sws_0085】．これは，あらかじめ型宣言が提供されているデータ型の再定義を避けるためである．

3.3.3　配列実装データ型

データ型名	<name>【rte_sws_7110】
概要	RTE API で使用するデータ型
備考	<name>は，配列実装データ型のショートネーム

配列実装データ型を提供するヘッダ

配列実装データ型は，RTE タイプヘッダ内に定義する〔rte_sws_7110〕．

配列実装データ型の設定

配列実装データ型は，*カテゴリ(category)*が ARRAY の*実装データ型(ImplementationDataType)*として定義する．

*実装データ型要素(ImplementationDataTypeElement)*の*ベース型(BaseType)*が*ネイティブ宣言(nativeDeclaration)*を持つ場合，RTE タイプヘッダは以下の型宣言を含む〔rte_sws_7110〕．

```
typedef <nativeDeclaration> <name>[<size 1>];
```

ここで，

- <nativeDeclaration>は，*ベース型(BaseType)*の*ネイティブ宣言(nativeDeclaration)*を表す．
- <name>は，配列実装データ型のショートネームを表す．
- <size 1>は，*実装データ型要素(ImplementationDataTypeElement)*の*配列サイズ(arraysize)*を表す．*配列サイズ(arraysize)*は 1 以上でなければならない【rte_sws_ext_1190】．

*カテゴリ(category)*が TYPE_REFERENCE の*実装データ型要素(ImplementationDataTypeElement)*で，その*実装データ型 (ImplementationDataType)* のカテゴリ*(category)*が VALUE の場合， RTE タイプヘッダは以下の型宣言を含む【rte_sws_7111】．

```
typedef <type> <name>[<size 1>];
```

ここで，

- <type>は，*実装データ型 (ImplementationDataType)* のショートネームを表す．
- <name>は，配列実装データ型のショートネームを表す．
- <size 1>は，*実装データ型要素(ImplementationDataTypeElement)*の*配列サイズ(arraysize)*を表す．*配列サイズ(arraysize)*は 1 以上でなければならない〔rte_sws_ext_1190〕．

AUTOSAR 仕様との違い

AUTOSAR 仕様では，*実装データ型要素(ImplementationDataTypeElement)* として，プリミティブ実装データ型以外を定義できる【rte_sws_7111】【rte_sws_6706】【rte_sws_6707】【rte_sws_6708】．しかし，本 RTE は配列の*実装データ型要素(ImplementationDataTypeElement)* としては，プリミティブ実装データ型のみサポートする【nrte_sws_0241】．

同一のデータ型の再定義の防止

ショートネーム，およびベース型(*BaseType*)のネイティブ宣言(*nativeDeclaration*)が等しい複数の配列実装データ型が定義されている場合，RTE タイプヘッダは〔rte_sws_7110〕に従った型宣言を 1 度だけ含む【rte_sws_7112】.

配列実装データ型のショートネーム，および*実装データ型 (ImplementationDataType)* のショートネームが等しい複数の配列実装データ型が定義されている場合，RTE タイプヘッダは〔rte_sws_7111〕に従った型宣言を 1 度だけ含む【rte_sws_7113】.

3.3.4 構造体実装データ型

データ型名	<name>【rte_sws_7114】
概要	RTE API で使用するデータ型
備考	<name>は，構造体実装データ型のショートネーム

構造体実装データ型を提供するヘッダ

構造体実装データ型は，RTE タイプヘッダ内に定義する〔rte_sws_7114〕.

構造体実装データ型の設定

構造体実装データ型は，*カテゴリ(category)*が STRUCTURE の*実装データ型 (ImplementationDataType)*として定義する.

構造体実装データ型に対して，RTE タイプヘッダは以下の型宣言を含む〔rte_sws_7114〕.

```
typedef struct { <elements> } <name>;
```

ここで,

- <elements>は，構造体要素の集合を表す．構造体要素の数は 1 以上でなければならない【rte_sws_ext_1192】.
- <name>は，構造体実装データ型のショートネームを表す.

構造体要素の集合

構造体要素の集合は，構造体実装データ型の*実装データ型要素(ImplementationDataTypeElement)*の集合に対応し，構造体要素の並び順は*実装データ型要素(ImplementationDataTypeElement)*の定義順である〔rte_sws_7114〕.

*実装データ型要素(ImplementationDataTypeElement)*の*カテゴリ(category)*が VALUE で，*ベース型(BaseType)*が*ネイティブ宣言(nativeDeclaration)*を持つ場合の構造体要素の出力形式は以下の通りである【rte_sws_7115】.

```
<nativeDeclaration> <name>;
```

ここで,

- <nativeDeclaration>は，*ベース型(BaseType)*の*ネイティブ宣言(nativeDeclaration)*を表す.
- <name>は，*実装データ型要素(ImplementationDataTypeElement)*のショートネームを表す.

*カテゴリ(category)*が TYPE_REFERENCE の*実装データ型要素(ImplementationDataTypeElement)*で，その*実装データ型（ImplementationDataType）*の*カテゴリ(category)*が VALUE の場合， RTE タイプヘッダは以下の型宣言を含む【rte_sws_7116】.

```
<type> <name>;
```

ここで,

- <type>は，*実装データ型（ImplementationDataType）*のショートネームを表す.
- <name>は，*実装データ型要素(ImplementationDataTypeElement)*のショートネームを表す.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，*実装データ型要素(ImplementationDataTypeElement)*の*カテゴリ(category)*にプリミティブ実装型以外を定義できる【rte_sws_7116】【rte_sws_7117】【rte_sws_7118】【rte_sws_7119】【rte_sws_7145】【rte_sws_7146】．しかし，本 RTE は構造体の*実装データ型要素(ImplementationDataTypeElement)*としては，プリミティブ実装データ型のみサポートする【nrte_sws_0243】.

同一のデータ型の再定義の防止

ショートネームが等しい複数の構造体実装データ型が定義されている場合，RTE タイプヘッダは〔rte_sws_7114〕に従った型宣言を 1 度だけ含む【rte_sws_7107】.

3.3.5 共用体実装データ型

データ型名	<name>【rte_sws_7144】
概要	RTE API で使用するデータ型
備考	<name>は，共用体実装データ型のショートネーム

共用体実装データ型を提供するヘッダ

共用体実装データ型は，RTE タイプヘッダ内に定義する〔rte_sws_7144〕.

共用体実装データ型の設定

共用体実装データ型は，*カテゴリ(category)*が UNION の*実装データ型(ImplementationDataType)*として定義する.

共用体実装データ型に対して，RTE タイプヘッダは以下の型宣言を含む〔rte_sws_7144〕.

```
typedef union { <elements> } <name>;
```

ここで,

- <elements>は，共用体要素の集合を表す．共用体要素の数は 2 以上でなければならない【rte_sws_ext_7147】.
- <name>は，共用体実装データ型のショートネームを表す.

共用体要素の集合

共用体要素の集合は，共用体実装データ型の*実装データ型要素(ImplementationDataTypeElement)*の集合に対応し，共用体要素の並び順は*実装データ型要素(ImplementationDataTypeElement)*の定義順である〔rte_sws_7144〕.

*実装データ型要素(ImplementationDataTypeElement)*の*カテゴリ(category)*が VALUE で，*ベース型(BaseType)*が*ネイティブ宣言(nativeDeclaration)*を持つ場合の共用体要素の出力形式は以下の通りである〔rte_sws_7115〕.

<nativeDeclaration> <name>;

ここで,

- <nativeDeclaration>は，*ベース型(BaseType)*の*ネイティブ宣言(nativeDeclaration)*を表す.
- <name>は，*実装データ型要素(ImplementationDataTypeElement)*のショートネームを表す.

*カテゴリ(category)*が TYPE_REFERENCE の*実装データ型要素(ImplementationDataTypeElement)*で，その*実装データ型 (ImplementationDataType)* の*カテゴリ(category)*が VALUE の場合， RTE タイプヘッダは以下の型宣言を含む〔rte_sws_7116〕.

<type> <name>;

ここで,

- <type>は，*実装データ型 (ImplementationDataType)* のショートネームを表す.
- <name>は，*実装データ型要素(ImplementationDataTypeElement)*のショートネームを表す.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，*実装データ型要素(ImplementationDataTypeElement)* の*カテゴリ(category)*にプリミティブ実装データ型以外を定義できる【rte_sws_7116】【rte_sws_7117】【rte_sws_7118】【rte_sws_7119】【rte_sws_7145】【rte_sws_7146】. しかし，本 RTE は共用体の*実装データ型要素(ImplementationDataTypeElement)* としては，プリミティブ実装データ型のみサポートする【nrte_sws_0245】.

同一のデータ型の再定義の防止

ショートネームが等しい複数の共用体実装データ型が定義されている場合，RTE タイプヘッダは〔rte_sws_7144〕に従った型宣言を 1 度だけ含む〔rte_sws_7107〕.

3.3.6　再定義実装データ型

データ型名	<name>【rte_sws_7109】
概要	RTE API で使用するデータ型
備考	<name>は，再定義実装データ型のショートネーム

実装データ型再定義を提供するヘッダ

再定義実装データ型は，RTE タイプヘッダ内に定義する〔rte_sws_7109〕．

実装データ型再定義の設定

再定義実装データ型は，*カテゴリ(category)*が TYPE_REFERENCE の*実装データ型(ImplementationDataType)*として定義する．

再定義実装データ型に対して，RTE タイプヘッダは以下の型宣言を含む〔rte_sws_7109〕．

```
typedef <type> <name>;
```

ここで，

- <type>は，再定義実装データ型が参照する実装データ型*(ImplementationDataType)*のショートネームを表す．
- <name>は，再定義実装データ型のショートネームを表す．

同一のデータ型の再定義の防止

再定義型実装データ型のショートネーム，および再定義型と互換する実装データ型のショートネームが等しい複数のプリミティブ実装データ型が定義されている場合，RTE タイプヘッダは〔rte_sws_7109〕に従った型宣言を 1 度だけ含む【rte_sws_7167】．

3.3.7　ポインタ実装データ型

データ型名	<name>【rte_sws_7148】
概要	RTE API で使用するデータ型
備考	<name>は，ポインタ実装データ型のショートネーム

ポインタ実装データ型を提供するヘッダ

ポインタ実装データ型は，RTE タイプヘッダ内に定義する〔rte_sws_7148〕．

ポインタ実装データ型の設定

ポインタ実装データ型は，*カテゴリ(category)*が DATA_REFERENCE の*実装データ型(ImplementationDataType)*として定義する．

ポインタ実装データ型に対して，RTE タイプヘッダは以下の型宣言を含む〔rte_sws_7148〕．

```
typedef <type> * <name>;
```

ここで，

- <name>は，ポインタ実装データ型のショートネームを表す．
- <type>は，ポインタ実装データ型の*ターゲットカテゴリ(targetCategory)*が VALUE の場合，*ベース型(BaseType)*の*ネイティブ宣言(nativeDeclaration)*を表す【rte_sws_7162】．ポインタ実装データ型の*ターゲットカテゴリ(targetCategory)*が TYPE_REFERENCE の場合，*実装データ型(ImplementationDataType)*のショートネームを表す【rte_sws_7163】．

AUTOSAR 仕様との違い

AUTOSAR 仕様では，ポインタ実装データ型として，型修飾子を付加することができる【rte_sws_7149】【rte_sws_7166】【rte_sws_7036】【rte_sws_7037】．しかし，本 RTE はポインタ実装データ型の型修飾子はサポートしない【nrte_sws_0311】.

同一のデータ型の再定義の防止

ショートネーム，およびその C 言語のポインタ型宣言が等しい複数のポインタ実装データ型が定義されている場合，RTE タイプヘッダは〔rte_sws_7148〕に従った型宣言を 1 度だけ含む【rte_sws_7169】.

3.3.8　モードのデータ型

データ型名	Rte_ModeType_<ModeDeclarationGroup>【rte_sws_7292】
概要	モード連携の API で使用するデータ型
備考	< ModeDeclarationGroup>は，モード宣言グループのショートネーム

モードのデータ型を提供するヘッダ(SCHM)

モードのデータ型は，モジュール連結タイプヘッダ内に定義する〔rte_sws_7292〕.

モードのデータ型の設定

各モード宣言グループに対して，モード連結タイプヘッダは以下の型宣言を含む〔rte_sws_7292〕.

```
#ifndef RTE_MODETYPE_<ModeDeclarationGroup>
#define RTE_MODETYPE_<ModeDeclarationGroup>
typedef <type> Rte_ModeType_<ModeDeclarationGroup>;
#endif
```

ここで，

- <ModeDeclarationGroup>は，*モード宣言グループ(ModeDeclarationGroup)* のショートネームを表す.
- <type>は，*モード要求型マップ (ModeRequestTypeMap)* の*実装データ型(implementationDataType)*のショートネームを表す．*実装データ型(implementationDataType)* は，プリミティブ実装データ型または再定義実装データ型(参照する実装データ型がプリミティブ実装データ型)である【rte_sws_a_0066】.

モード遷移中のステータスの定義

各モード宣言グループに対するモード遷移中のステータスとして，モード連結タイプヘッダは以下の定義を含む【rte_sws_7293】.

```
#ifndef RTE_TRANSITION_<ModeDeclarationGroup>
#define RTE_TRANSITION_<ModeDeclarationGroup> ¥
 <n>U
#endif
```

ここで,

- <ModeDeclarationGroup>は，*モード宣言グループ(ModeDeclarationGroup)* のショートネームを表す.
- <n>は，モード宣言グループに含まれるモード宣言の数を表す.

モード宣言の定義

モード宣言グループに含まれるモード宣言毎に，モード連結タイプヘッダは以下の定義を含む【rte_sws_7294】.

```
#ifndef RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration>
#define RTE_MODE_<ModeDeclarationGroup>_<ModeDeclaration> ¥
 <index>U
#endif
```

ここで,

- <ModeDeclarationGroup>は，*モード宣言グループ(ModeDeclarationGroup)* のショートネームを表す.
- <ModeDeclaration>は，*モード宣言 (ModeDeclaration)* のショートネームを表す.
- <index>は，モード宣言グループに含まれるモード宣言をアルファベット順で並べたときのインデックス番号(最小値は 0)を表す.

3.4　マクロ

3.4.1　API 返り値チェック

Std_ReturnType の各エラービットを直接参照するのを避けるために，RTE は以下の 3 つのマクロを提供する．SW-C はこれらのマクロを使用して API の返り値をチェックすることができる．

Rte_IsInfrastructureError

マクロ名	Rte_IsInfrastructureError(status)【rte_sws_7404】
概要	API 返り値に，即時通信基盤エラーフラグのビットがセットされたときに 0 以外の値を返す.

Rte_HasOverlayedError

マクロ名	Rte_HasOverlayedError(status)【rte_sws_7405】
概要	API 返り値に，オーバーレイエラーフラグのビットがセットされたときに 0 以外の値を返す.

Rte_ApplicationError

マクロ名	Rte_ApplicationError(status)【rte_sws_7406】
概要	API 返り値のアプリケーションエラーコード(下位 6 ビット)を返す.

3.5　定数

3.5.1　初期値定数

定数名	Rte_InitValue_<Port>_<DEPType>【rte_sws_5078】
概要	*実装データ型(ImplementationDataType)*のカテゴリ *(category)*が VALUE のデータ要素の初期値【rte_sws_5078】
備考	<Port>は S/R 連携の送信側，もしくは受信側ポートのショートネーム． <DEPType>は S/R 連携の送信側，もしくは受信側データ要素のショートネーム．

初期値定数を提供するヘッダ

初期値定数は，アプリケーションヘッダ内に定義する【rte_sws_5078】．

初期値定数の設定

RTEGEN は，データセマンティクスの送信側，もしくは受信側データ要素毎に，初期値定数を生成する【rte_sws_5078】.

```
#define Rte_InitValue_<Port>_<DEPType> ((<DataType>) <initValue>)
```

ここで

- <Port>は，送信側，もしくは受信側データ要素の所属するポートのショートネーム.
- <DEPType>は，データ要素のショートネーム.
- <DataType>は，データ要素の *実装データ型(ImplementationDataType)*のショートネーム.
- <initValue>は，データの初期値.

3.5.2　アプリケーションエラー定数

定数名	RTE_E_<interface>_<error>【rte_sws_2576】
概要	アプリケーションエラーの定数値
備考	<interface>は C/S インタフェースのショートネーム， <error>はアプリケーションエラーのショートネーム.

アプリケーションエラー定数を提供するヘッダ

アプリケーションエラー定数は，アプリケーションヘッダで定義される【rte_sws_2575】．

アプリケーションエラー定数の設定

アプリケーションエラー定数は以下のように定義する【rte_sws_2576】．

```
#define RTE_E_<interface>_<error> ((Std_ReturnType)<error value>)
```

ここで，

- <interface>は，C/S インタフェースのショートネーム.

・ <error>は，アプリケーションエラーのショートネーム．

アプリケーションエラー定数の多重定義の防止

RTEGEN は，〔rte_sws_2575〕中の<interface>，<error>，および<error value>の等しいアプリケーションエラー定数を 1 度だけ生成する【rte_sws_7143】．

3.5.3　列挙定数

列挙定数は，データ型のとりうる値を示す定数をアプリケーションに提供する．

定数名	<prefix><EnumLiteral>【rte_sws_3810】
概要	データ型の列挙値
備考	<EnumLiteral>は，*計算スケール(CompuScale)の計算定数(CompuConst)の文字列値(vt)*. <prefix>は，*計算方式(CompuMethod)*を使用する *AUTOSAR データ型(AutosarDataType)*を参照する*インクルードデータ型セット(IncludedDataTypeSet)の リテラル接頭辞(literalPrefix)*.

列挙定数を提供するヘッダ

SW-C で使用する*実装データ型(ImplementationDataType)*に対する全ての列挙定数は，アプリケーションタイプヘッダに定義する【rte_sws_3809】．

同じ*実装データ型(ImplementationDataType)*，あるいは同じ*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*が，異なる*リテラル接頭辞(literalPrefix)*の*インクルードデータ型セット(IncludedDataTypeSets)*から参照される場合，列挙定数は*リテラル接頭辞(literalPrefix)*毎に定義する【rte_sws_8401】．

列挙定数の設定

*計算方式(CompuMethod)*が以下の条件を全て満たす場合，*計算方式(CompuMethod)の内部値-物理値計算(compuInternalToPhys)の計算スケール(CompuScale)*毎に，RTEGEN は，以下の定義を生成する【rte_sws_3810】．

・ 条件 1）〔rte_sws_3809〕の*実装データ型(ImplementationDataType)*から参照されている．
・ 条件 2）*カテゴリ(category)*が TEXTTABLE.
・ 条件 3）点範囲を表す*計算スケール(CompuScale)*のみを含む．

```
#ifndef <prefix><EnumLiteral>
#define <prefix><EnumLiteral> ((<type>) <value>)
#endif /* <prefix><EnumLiteral> */
```

ここで，

・ <EnumLiteral>は，*計算スケール(CompuScale)の計算定数(CompuConst)の文字列値(vt)*.

- <prefix>は，*計算方式(CompuMethod)*を使用する *AUTOSAR データ型(AutosarDataType)*を参照する*インクルードデータ型セット(IncludedDataTypeSet)*の*リテラル接頭辞(literalPrefix)*.
- <type>は，*計算方式(CompuMethod)*が属する*実装データ型(ImplementationDataType)*の識別子.
- <value>は，*計算スケール(CompuScale)*の点範囲の値.

列挙定数名の<prefix>はオプションであり，シンボル名の重複を避けるために使用することができる.

3.5.4　上限値，および下限値定数

定数名	<prefix><DataType>_LowerLimit <prefix><DataType>_UpperLimit【rte_sws_5052】
概要	データ型の上限値，および下限値
備考	<DataType>は，*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*のショートネーム. <prefix>は，*データ制約(dataConstr)*が属する *AUTOSAR データ型(AutosarDataType)*を参照する*インクルードデータ型セット(IncludedDataTypeSet)*の*リテラル接頭辞(literalPrefix)*

範囲データ型は，とりうる値の範囲の上限値，および下限値を示す定数を SW-C に提供する.

上限値，および下限値定数を提供するヘッダ

SW-C で使用する全ての*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*毎に，上限値，および下限値定数の定義を，アプリケーションタイプヘッダに定義する【rte_sws_5051】.

同じ*実装データ型(ImplementationDataType)*，あるいは同じ*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*が異なる*インクルードデータ型セット(IncludedDataTypeSets)*から参照される場合，異なる*リテラル接頭辞(literalPrefix)*と*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*の組み合わせ毎に，下限値，および上限値定数を定義する【rte_sws_8402】.

上限値，および下限値定数のコンフィギュレーション

*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*に対して，*下限値(lowerLimit)*，および*上限値(upperLimit)*を指定した*データ制約(dataConstr)*を設定することで，データ型の上限値，および下限値を指定できる.

下限値(lowerLimit)，および*上限値(upperLimit)*を指定した*データ制約(dataConstr)*を参照する*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*に対して，RTEGEN は，アプリケーションタイプヘッダに以下の定義を生成する【rte_sws_5052】.

```
#define <prefix><DataType>_LowerLimit ((<ImplType>) <lowerValue>)
```

```
#define <prefix><DataType>_UpperLimit ((<ImplType>) <upperValue>)
```

ここで，

- <DataType>は，*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*のショートネーム．
- <ImplType>は，*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*にマッピングされた*実装データ型(ImplementationDataType)*のショートネーム．
- <prefix>は，*データ制約(dataConstr)*が属する *AUTOSAR データ型(AutosarDataType)*を参照する*インクルードデータ型セット(IncludedDataTypeSet)*の *リテラル接頭辞(literalPrefix)*．
- <lowerValue>は，*データ制約(dataConstr)*の *下限値(lowerLimit)*の値．
- <upperValue>は，*データ制約(dataConstr)*の *上限値(upperLimit)*の値．

なお<prefix>について，*AUTOSAR データ型(AutosarDataType)*は，*アプリケーションデータ型(ApplicationDataType)*，もしくは*アプリケーションデータ型(ApplicationDataType)*にマッピングされた*実装データ型(ImplementationDataType)*のいずれかを示す．

3.6　ランナブル

ランナブルのエントリポイント関数は，SW-C のソースコード内で定義された関数として実装される．以降の節ではこの関数のシグネチャやプロトタイプについて記載する．

3.6.1　ランナブルのシグネチャ

ランナブルのシグネチャは，起動契機となる RTE イベントの種類に関係なく以下の形式となる【rte_sws_1126】．

```
<void|Std_ReturnType> <name> ( [role parameters] )
```

ここで，

- <name>はランナブルのエントリポイント関数のシンボル名．
- <role parameters>はロールパラメータ．ランナブルのロールパラメータの詳細は 3.6.3 節を参照．

3.6.2　エントリポイント関数

RTEGEN は，ランナブルのエントリポイント関数のプロトタイプを，メモリマッピングマクロとコンパイラ抽象化マクロにより，以下のようにラップしてアプリケーションヘッダに生成する【rte_sws_7194】．

```
#define <c>_START_SEC_<sadm>
#include "<c>_MemMap.h"

<void|Std_ReturnType> <name> ([role parameters]);

#define <c>_STOP_SEC_<sadm>
#include "<c>_MemMap.h"
```

ここで

- <c>は，SW-C 型のショートネーム．
- <sadm>は，ランナブルが参照する*ソフトウェアアドレッシング方式(SwAddrMethod)*のショートネーム．
- <name>は，ランナブルの*エントリポイントシンボル名(symbol)*．
- <role paramerters>は，ランナブルのロールパラメータ．ロールパラメータの詳細は 3.6.3 節を参照．
- ランナブルが*ソフトウェアアドレッシング方式(SwAddrMethod)*を参照していない場合は<sadm>はデフォルトの"CODE"とする．

複数のランナブルが同じ*ソフトウェアアドレッシング方式(SwAddrMethod)*を参照している場合，メモリマッピングマクロは，複数のエントリポイント関数のプロトタイプをラップできる．

ランナブルのエントリポイント関数を提供するヘッダ

RTEGEN は，ランナブルのエントリポイント関数のプロトタイプをアプリケーションヘッダに生成する【rte_sws_1132】.

サポート範囲の制限

AUTOSAR 仕様では，以下のように<name>の前に<prefix>を指定してエントリポイント関数のプロトタイプを生成可能とされているが，本 RTE では，<prefix>の指定をサポートしない【nrte_sws_0062】.

<void|Std_ReturnType> **<prefix>**<name> ([role parameters]);

3.6.3　ロールパラメータ

RTE は，ロールパラメータの各パラメータの存在の有無，およびパラメータの型をランナブルの起動契機となる RTE イベントから決定する.

各 RTE イベントのロールパラメータの詳細は，3.6.5 節を参照.

3.6.4　返り値

RTE は，C/S 連携のオペレーション呼出しのエラーコードをアプリケーションに返すためにのみ，ランナブルの返り値を使用する.

ランナブルがサーバランナブルであり，C/S インタフェースにおいて，使用アプリケーションエラーが定義されている場合のみ，ランナブルのエントリポイント関数の返り値は，Std_ReturnType となり，それ以外の場合，エントリポイント関数の返り値は void である【rte_sws_1130】.

サーバランナブルは，返り値の下位 6 ビットのみをアプリケーションエラーに使用し，上位 2 ビットは"0"に設定して，返り値を返さなければならない．【rte_sws_ext_2704】.

3.6.5　ランナブルを起動するイベント

RTE は，RTE イベントに対応したランナブルを起動する．以降の節では，各 RTE イベントについて，ランナブルを起動する条件について説明する.

各イベントに対して，ランナブルのエントリポイント関数のシグネチャが定義されている．エントリポイント関数のシグネチャは以下のパラメータを含む.

・　ロールパラメータ

トリガ条件の結果の情報を渡すのに使用する．ロールパラメータの種別，多重度，および順序は起動契機の RTE イベントに依存する.

3.6.5.1　周期イベント

C 言語 I/F	void <name>(void)　【rte_sws_1131】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
機能	
コンフィギュレーションで定義された起動周期で，ランナブルを周期起動する．	

3.6.5.2　バックグラウンドイベント

C 言語 I/F	void <name>(void)　【rte_sws_7177】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
機能	
バックグラウンド動作実行に利用される循環型イベント．周期イベントと同様だが，固定周期を持たず，低優先度でのみ起動される．	

3.6.5.3　データ送信完了イベント

C 言語 I/F	void <name>(void)　【rte_sws_1137】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
機能	
送信 ACK 発生により，ランナブルを起動する．	

3.6.5.4　データ受信イベント

C 言語 I/F	void <name>(void)　【rte_sws_1135】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
機能	
データ受信時に，ランナブルを起動する．	

3.6.5.5　データ受信エラーイベント

C 言語 I/F	void <name>(void)　【rte_sws_1359】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
機能	
データ要素のエラー状態検出時に，ランナブルを起動する．	

3.6.5.6　オペレーション呼出しイベント

C 言語 I/F	<void | Std_ReturnType> <name> ([IN <portDefArg 1>, ..., <portDefArg n>], [IN|INOUT|OUT <param 1>, ..., IN|INOUT|OUT <param n>]) 【rte_sws_1166】	
パラメータ[in]	<portDefArg 1>, ..., <portDefArg n>	ポート定義引数
パラメータ[in/out]	<param 1>, ..., <param n>]	オペレーション引数
パラメータ[out]	－	
返り値	アプリケーションエラー	
機能		
サーバランナブルの起動イベント．このイベントは，サーバがクライアントからオペレーション呼出しを受信した際に発生する． <portDefArg 1>, …, <portDefArg n>は，ポート定義引数 <param1>, …, <param n>はオペレーション引数		

<u>パラメータの仕様</u>

ポート定義引数のデータ型は，ポート定義引数の*ポート定義引数値型(valueType)*で定義される．

ポート定義引数はオプションである．

オペレーション引数<param 1>, …, <param n>は，*オペレーション呼出しイベント(OperationInvokedEvent)*に関連するオペレーションのオペレーション引数で定義される．

オペレーション引数の順序は，オペレーション内のオペレーション引数の定義順に従う【rte_sws_7023】．

オペレーション引数の*サーバ引数実装ポリシ(ServerArgumentImplPolicy)*が useArgumentType に設定されている場合，<param>のデータ型は，オペレーション引数の*実装データ型(ImplementationDataType)*で定義される【rte_sws_7024】．本 RTE では，*サーバ引数実装ポリシ(ServerArgumentImplPolicy)*は常に useArgumentType に設定されているものとして扱い，*サーバ引数実装ポリシ(ServerArgumentImplPolicy)*のコンフィギュレーションはサポートしない【nrte_sws_0063】．

同一のランナブルに対して，複数の*オペレーション呼出しイベント(OperationInvokedEvent)*が設定されることでシグニチャが異なる場合，RTEGEN は設定を拒否する【rte_sws_3526】．本 RTEGEN では，生成コードがコンパイル時にエラー検出されることをもって，本仕様に対応したこととする．

<u>返り値の仕様</u>

C/S インタフェースにアプリケーションエラーが定義されている場合，これらのエラーは Std_ReturnType 型の返り値で返される．アプリケーションエラーの定義がない場合，返り値は void になる．

RTE_E_OK を常に返すようなサーバランナブルであっても，ランナブルの返り値を Std_ReturnType とするため，アプリケーションエラーコードの定義は必要である．

3.6.6　リエントラント性

ランナブルのリエントラント性を保証するのは SW-C の実装者の責任である．

3.7　BSW スケジューラブル

BSW スケジューラブルのエントリポイント関数は，BSWM のソースコード内で定義された関数として実装される．以降の節ではこの関数のシグネチャやプロトタイプについて記載する．

3.7.1　BSW スケジューラブルのシグネチャ

BSW スケジューラブルのシグネチャは，起動契機となる BSW イベントの種類に関係なく以下の形式となる【rte_sws_7282】.

void <bsnp>_<name>(void)

ここで,

- <bsnp> は，BSWM ディスクリプションのショートネーム〔rte_sws_7593〕.
- <name>は BSW スケジューラブルの *BSWM エントリ(BswModuleEntry)*のショートネームの”<bsnp>_”以降の名称.

BSW スケジューラブルの *BSWM エントリ(BswModuleEntry)*は引数と返り値を持たないこと【rte_sws_ext_7287】.

AUTOSAR 仕様との違い

AUTOSAR 仕様では BSWM の *ベンダ ID(vendorId)*と *ベンダ API 接中辞(vendorApiInfix)*を接頭辞に含めている【rte_sws_7282】.

しかし, A-RTEGEN では, SCHM のマルチインスタンスはサポートしないため, *ベンダ ID(vendorId)* と *ベンダ API 接中辞(vendorApiInfix)*は使用しない〔irte_sws_0009〕.

3.7.2　エントリポイント関数

RTEGEN は, BSW スケジューラブルのエントリポイント関数のプロトタイプを，メモリマッピングマクロとコンパイラ抽象化マクロにより，以下のようにラップしてアプリケーションヘッダに生成する【rte_sws_7195】.

```
#define <snp>_START_SEC_<sadm>
#include "MemMap.h"
<void|Std_ReturnType> <bsnp>_<name> (void);
#define <snp>_STOP_SEC_<sadm>
#include "MemMap.h"
```

ここで,

- <snp>は，BSWM ディスクリプションのショートネーム(大文字)【rte_sws_7595】
- <bsnp> は，BSWM ディスクリプションのショートネーム〔rte_sws_7593〕.
- <name>は BSW スケジューラブルの *BSWM エントリ(BswModuleEntry)*のショートネーム.
- <sadm>は，BSW スケジューラブルが参照する *ソフトウェアアドレッシング方式(SwAddrMethod)*のショートネーム.

・ BSW スケジューラブルが *ソフトウェアアドレッシング方式(SwAddrMethod)*を参照していない場合は<sadm>はデフォルトの"CODE"とする.

複数の BSW スケジューラブルが同じ *ソフトウェアアドレッシング方式(SwAddrMethod)*を参照している場合，メモリマッピングマクロは，複数のエントリポイント関数のプロトタイプをラップできる.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，メモリマッピングマクロ名に，BSWM の *ベンダ ID(vendorId)*と *ベンダ API 接中辞(vendorApiInfix)*を接頭辞に含めている【rte_sws_7195】.

しかし, A-RTEGEN では, SCHM のマルチインスタンスはサポートしないため, *ベンダ ID(vendorId)* と *ベンダ API 接中辞(vendorApiInfix)*は使用しない〔irte_sws_0009〕.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，*BSWM エンティティ(BswModuleEntity)*に*セクション名接頭辞(SectionNamePrefix)*を指定している場合，<snp>は*セクション名接頭辞(SectionNamePrefix)*のショートネームとしている【rte_sws_7596】.

しかし，A-RTEGEN では，*セクション名接頭辞(SectionNamePrefix)*はサポートしない【irte_sws_0010】.

BSW スケジューラブルのエントリポイント関数を提供するヘッダ

RTEGEN は，各 BSW スケジュラブルのエントリポイント関数のプロトタイプをモジュール連結ヘッダファイルに生成する【rte_sws_7283】.

3.7.3　リエントラント性

BSW スケジューラブルのリエントラント性を保証するのは BSWM の実装者の責任である【nrte_sws_ext_0019】.

3.8 RTE API

3.8.1 Rte_Write

C 言語 I/F	Std_ReturnType Rte_Write_<p>_<o>(IN　<data>)【rte_sws_1071】	
パラメータ[in]	<data>	送信するデータ
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	RTE_E_OK	通信サービスへ正常にデータが渡された【rte_sws_7820】.
	RTE_E_COM_STOPPED (ECU 間連携の場合)	COM が現在使用できないため，RTE が処理を実行できなかった．COM が COM_SERVICE_NOT_AVAILABLE を返したときはこの値を返す【rte_sws_7822】.
	RTE_E_SEG_FAULT	RTE API に渡されるパラメータに対して，〔rte_sws_2752〕，〔rte_sws_2753〕のセグメンテーション違反を検出したときは，データ伝搬せずに，この値を返す【rte_sws_2756】
	RTE_E_DEV_DEFECT	A-RTE がデータ伝搬で使用する OS/IOC/COM のシステムサービスにおいて，拡張エラーが発生したときはこの値を返す【irte_sws_0015】.
同期/非同期	ECU 内連携の場合，同期．ECU 間連携の場合，非同期.	
リエントラント	○	
生成有無の設定	データセマンティクスの送信側データ要素に対して，*データ送信ポイント(dataSendPoint)*ロールの*変数アクセス(VariableAccess)*が存在する場合，本 API を生成する【rte_sws_1280】.	
機能		
データセマンティクスの S/R 連携のデータ送信を開始する. ・ <p>は送信側ポートのショートネーム. ・ <o>は送信側データ要素のショートネーム. ・ <data>は送信するデータであり，データ型は送信側データ要素の*実装データ型(ImplementationDataType)*.		

API 呼出しの制限

本 API は，*データ送信ポイント(dataSendPoint)*ロールの*変数アクセス(VariableAccess)*を持つランナブルだけが使用可能である【rte_sws_ext_7818】.

<data>が参照渡しである場合，そのポインタは本 API が復帰するまで有効でなければならない【rte_sws_ext_a_0010】.

機能仕様

本 API は，API 呼び出しが行われた時点で S/R 連携のデータ送信を開始する．API の返り値は，本 API 処理中に RTE が検出したエラーを示す．1:N 連携において，複数のエラーを検出した場合の API の返り値は実装定義とする【nrte_sws_0146】．A-RTE では，最後に検出したエラーを API の返り値とする【irte_sws_0002】.

本 API は，有限の時間内に終了する【rte_sws_6016】.

ECU 間連携の場合，本 API は即時に送信要求を行う【rte_sws_7824】．そして，データを通信サービスに渡した時点で復帰する【rte_sws_7826】．本 API 呼び出しによる送信要求が実際の送信となるわけではない．実際の送信が行われるかどうかは RTE，および通信サービスのコンフィギュレーションに依存する.

ECU 内連携の場合，本 API は，RTE 内部の受信データセット，もしくは IOC バッファにデータをコピーした後に復帰する【rte_sws_2635】.

未接続の送信側ポートに対応する Rte_Send/Rte_Write は，何も行わず，返り値として RTE_E_OK を返す【rte_sws_1332】.

3.8.2 Rte_Send

C 言語 I/F	Std_ReturnType Rte_Send_<p>_<o> (IN <data>)【rte_sws_1072】	
パラメータ[in]	<data>	送信するデータ
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	RTE_E_OK	通信サービスへ正常にデータが渡された【rte_sws_7821】.
	RTE_E_COM_STOPPED (ECU 間連携の場合)	COM が現在使用できないため，RTE が処理を実行できなかった．COM が COM_SERVICE_NOT_AVAILABLE を返したときはこの値を返す【rte_sws_7823】.
	RTE_E_LIMIT (ECU 内連携の場合)	キューが一杯になり，ECU 内のローカルな受信者の 1 つによってイベントが捨てられた【rte_sws_2634】.
	RTE_E_SEG_FAULT	RTE API に渡されるパラメータに対して，〔rte_sws_2752〕，〔rte_sws_2753〕のセグメンテーション違反を検出したときは，データ伝搬せずに，この値を返す【rte_sws_2754】
	RTE_E_DEV_DEFECT	A-RTE がデータ伝搬で使用する OS/IOC/COM のシステムサービスにおいて，拡張エラーが発生したときはこの値を返す【irte_sws_0016】.
同期/非同期	非同期	
リエントラント	○	
生成有無の設定	イベントセマンティクスの送信側データ要素に対して，*データ送信ポイント(dataSendPoint)*ロールの*変数アクセス(VariableAccess)*が存在する場合，本 API を生成する【rte_sws_1281】.	
機能		
イベントセマンティクスにおける S/R 連携のデータ送信を開始する. ・ <p>は送信側ポートのショートネーム. ・ <o>は送信側データ要素のショートネーム. ・ <data>は送信するデータであり，データ型は送信側データ要素の*実装データ型(ImplementationDataType)*.		

API 呼出しの制限

本 API は*データ送信ポイント(dataSendPoint)*ロールの*変数アクセス(VariableAccess)*を含むランナブルだけが使用する【rte_sws_ext_7819】.

<data>が参照渡しである場合，そのポインタは本 API が復帰するまで有効でなければならない【rte_sws_ext_a_0011】.

機能仕様

本 API は，API 呼び出しが行われた時点で S/R 連携のデータ送信を開始する．返り値は，本 API 処理中に RTE が検出したエラーを返す．1:N 連携において，複数のエラーを検出した場合の API の返り値は実装定義とする【nrte_sws_0146】．A-RTE では，最後に検出したエラーを API の返り値とする〔irte_sws_0002〕.

本 API は，有限の時間内に終了する【rte_sws_6016】.

ECU 間連携の場合，本 API は即時に送信要求を行う【rte_sws_7825】．そして，データを通信サービスに渡した時点で復帰する【rte_sws_7827】．送信要求が実際の送信となるわけではなく他の要因に依存するかどうかは，RTE，および通信サービスのコンフィギュレーションに依存する.

ECU 内連携の場合，本 API は RTE 内部の受信キュー，もしくは IOC バッファにデータをキューイングした後で復帰する【rte_sws_2633】.

S/R 連携の未接続の送信側ポートに対する Rte_Send は何も行わず，引数として RTE_E_OK を返す【rte_sws_1332】.

3.8.3　Rte_Invalidate

C 言語 I/F	Std_ReturnType Rte_Invalidate_<p>_<o> (void)【rte_sws_1206】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	RTE_E_OK	成功【rte_sws_1207】.
	RTE_E_COM_STOPPED (ECU 間通信の場合)	COM が現在使用できないため，RTE が処理を実行できなかった COM が COM_SERVICE_NOT_AVAILABLE を返したときはこの値を返す【rte_sws_1339】.
	RTE_E_DEV_DEFECT	A-RTE がデータ伝搬で使用する OS/IOC/COM のシステムサービスにおいて，拡張エラーが発生したときはこの値を返す【irte_sws_0017】.
同期/非同期	非同期	
リエントラント	○	
生成有無の設定	*無効化ポリシ(InvalidationPolicy)*が keep，もしくは replace である送信側データ*要素*を参照するデータ*送信ポイント(dataSendPoint)*ロールの*変数アクセス(VariableAccess)*が存在する場合，本 API を生成する【rte_sws_1282】.	
機能		
S/R 連携におけるデータ無効化を開始する. ・　<p>は送信側ポートのショートネーム. ・　<o>は送信側データ要素のショートネーム.		

API 呼出しの制限

本 API は，*データ送信ポイント(dataSendPoint)*ロールの*変数アクセス(VariableAccess)*を持つランナブルからのみ使用されなければならない【rte_sws_ext_2682】.

機能仕様

本 API は，API 呼出しが行われた時点で S/R 連携のデータ無効化を開始する．返り値は「成功」，あるいは，本 API 処理中に RTE が検出したエラーを返す.

S/R 連携の未接続の送信側ポートに対応する Rte_Invalidate は，何も行わず，返り値として RTE_E_OK を返す【rte_sws_3783】.

3.8.4　Rte_Feedback

C 言語 I/F	Std_ReturnType Rte_Feedback_<p>_<o>([IN Rte_Instance <instance>])【rte_sws_1083】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	RTE_E_NO_DATA	送信 ACK，もしくはエラー通知が COM から受信されていない【rte_sws_1084】.
	RTE_E_COM_STOPPED (ECU 間通信の場合)	Rte_Send，Rte_Write，もしくは Rte_Invalidate(※)により最後に行われた送信が戻り値 RTE_E_COM_STOPPED により拒否された【rte_sws_7636】. COM からタイムアウト通知を受信する前にエラー通知を受信した【rte_sws_3774】. (※)Rte_Invalidate も対象とする.
	RTE_E_TIMEOUT (ECU 間通信の場合)	COM からエラー通知を受信する前にタイムアウト通知を受信した【rte_sws_7637】.
	RTE_E_TRANSMIT_ACK	以下のいずれかの場合【rte_sws_1086】. ・ECU 間連携において，COM から送信 ACK の通知を受信した. ・ECU 内連携である.
	RTE_E_UNCONNECTED	送信ポートが未接続である【rte_sws_7658】.
	RTE_E_DEV_DEFECT	A-RTE がデータ伝搬で使用する OS/IOC/COM のシステムサービスにおいて，拡張エラーが発生したときはこの値を返す【irte_sws_0027】.
同期/非同期	非同期	
リエントラント	○	
生成有無の設定	提供側データ要素毎に，以下の全ての条件を満たす場合，Rte_Feedback API を生成する【rte_sws_1285】.	

	・送信 ACK が有効である. ・dataSendPoint ロールの VariableAccess が VariableDataPrototype を参照している.
機能	
S/R 連携における送信 ACK を取得する. ・　<p>は送信側ポートのショートネーム. ・　<o>は送信側データ要素のショートネーム.	

API 呼出しの制限

特になし

機能仕様

Rte_Feedback の呼び出しは，Rte_Feedback の戻り値として提供されるステータスを変更しない【rte_sws_7634】.

各シグナル, もしくはシグナルグループについて, Rte_Send, Rte_Write, もしくは Rte_Invalidate(※)の送信要求後の COM からの最初の通知のみを送信 ACK として受け付ける【rte_sws_7635】.

未接続の提供側ポートの Rte_Feedback API は，即座に RTE_E_UNCONNECTED を返す【rte_sws_1344】.

送信を行っていない初期状態においては，Rte_Feedback API の返り値は RTE_E_TRANSMIT_ACK である【rte_sws_7652】.

(※)Rte_Invalidate も対象とする.

3.8.5　Rte_Read

C 言語 I/F	Std_ReturnType Rte_Read_<p>_<o> (OUT <data>)【rte_sws_1091】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	<data>	受信したデータを格納する領域へのポインタ. データ受信前には，初期値を返す【rte_sws_7396】.
返り値	RTE_E_OK	データ読み取り成功【rte_sws_1093】.
	RTE_E_INVALID	データ要素が無効【rte_sws_2626】.
	RTE_E_MAX_AGE_EXCEEDED	データ要素が古い. このオーバーレイエラーはその他のエラーコードと結合しうる【rte_sws_2703】.
	RTE_E_UNCONNECTED	受信ポートが未接続【rte_sws_7690】.
	RTE_E_COM_STOPPED (ECU 間連携の場合)	COM が現在使用できないため，RTE が処理を実行できなかった．COM が COM_SERVICE_NOT_AVAILABLE を返したときはこの値を返す【nrte_sws_0237】.
	RTE_E_SEG_FAULT	RTE API に渡されるパラメータに対して，〔rte_sws_2752〕，〔rte_sws_2753〕のセグメンテーション違反を検出したときは，データ伝搬せずに，この値を返す【nrte_sws_0259】.
	RTE_E_DEV_DEFECT	A-RTE がデータ伝搬で使用する OS/IOC/COM のシステムサービスにおいて，拡張エラーが発生したときはこの値を返す【irte_sws_0018】.
同期/非同期	同期	
リエントラント	○	
生成有無の設定	*データセマンティクスの受信側データ要素に対し，引数によるデータ受信ポイント(dataReceivePointByArgument)ロールの変数アクセス*	

	*(VariableAccess)*が存在する場合，本 API を生成する【rte_sws_1289】.
機能	
データセマンティクスにおける S/R 連携のデータ取得を行う. ・ <p>は受信側ポートのショートネーム. ・ <o>は受信側データ要素のショートネーム. ・ <data>は取得したデータの格納先変数への参照であり，データ型は受信側データ要素の*実装データ型(ImplementationDataType)*への参照型.	

API 呼出しの制限

本 API は，引数による*データ受信ポイント(dataReceivePointByArgument)*ロールの*変数アクセス(VariableAccess)*を保持するランナブルからのみ使用されなければならない【rte_sws_ext_2683】.

<data>のポインタは本 API が復帰するまで有効でなければならない【rte_sws_ext_a_0004】.

機能仕様

本 API は，受信したデータの取得を行う．返り値は，本 API 処理中に RTE が検出したエラー，あるいは COM で検出したエラーを返す.

本 API は，有限の時間内に終了する【rte_sws_2519】.

S/R 連携の未接続の受信側ポートに対する Rte_Read は，返り値として RTE_E_UNCONNECTED を返し，<data>の出力値として初期値を提供する【rte_sws_1330】.

ノンブロッキング Rte_Read は，データ未受信かどうかを示す【rte_sws_6012】ために，出力値として初期値が返る.

3.8.6　Rte_Receive

C 言語 I/F	Std_ReturnType Rte_Receive_<p>_<o> (OUT <data>) 【rte_sws_1092】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	<data>	受信したデータを格納する領域へのポインタ
返り値	RTE_E_OK	データ読み取り成功 【rte_sws_2598】.
	RTE_E_NO_DATA	本 API 呼出し時に受信イベントも発生エラーもなかった 【rte_sws_1094】.
	RTE_E_LOST_DATA	受信キューのオーバーフロー，あるいは(データ到達順序エラーを含む)通信サービスのエラーのために，受信データの一部をロストした. これは，<data>で返されたデータのエラーではない. このオーバーレイエラーは他のエラーと結合しうる 【rte_sws_2572】.
	RTE_E_UNCONNECTED	受信ポートが未接続 【rte_sws_7665】.
	RTE_E_SEG_FAULT	RTE API に渡されるパラメータに対して，〔rte_sws_2752〕，〔rte_sws_2753〕のセグメンテーション違反を検出したときは，データ伝搬せずに，この値を返す【nrte_sws_0260】.
	RTE_E_DEV_DEFECT	A-RTE がデータ伝搬で使用する OS/IOC/COM のシステムサービスにおいて，拡張エラーが発生したときはこの値を返す 【irte_sws_0019】.
同期/非同期	同期	

リエントラント	○
生成有無の設定	*イベントセマンティクスの受信側データ要素に対し，引数によるデータ受信ポイント(dataReceivePointByArgument)ロールの変数アクセス(VariableAccess)*が存在する場合，本 API を生成する【rte_sws_1288】.
機能	
イベントセマンティクスにおける S/R 連携の受信データ取得を行う. ・ <p>は受信側ポートのショートネーム. ・ <o>は受信側データ要素のショートネーム. ・ <data>は取得したデータの格納先変数への参照であり，データ型は受信側データ要素の*実装データ型(ImplementationDataType)*への参照型.	

API 呼出しの制限

本 API は，引数による*データ受信ポイント(dataReceivePointByArgument)ロールの変数アクセス(VariableAccess)*を保持するランナブルからのみ使用されなければならない【rte_sws_ext_2684】.

<data>のポインタは，本 API が復帰するまで有効でなければならない【rte_sws_ext_a_0005】.

機能仕様

本 API は，受信したデータの取得(キューからの取り出し)を行う．本 API は本 API 処理中に RTE が検出したエラーあるいは COM で検出したエラーを返す．RTE_E_NO_DATA, RTE_E_UNCONNECTED はエラーではなく API 呼出しの正常な動作を示す.

返り値が RTE_E_NO_DATA，RTE_E_TIMEOUT，RTE_E_UNCONNECTED，もしくは RTE_E_IN_EXCLUSIVE_AREA の場合，<OUT>引数の参照先の値は変更しない【rte_sws_7673】.

S/R 連携の未接続の受信側ポートに対応する Rte_Receive は，何も行わず，即座に RTE_E_UNCONNECTED を返す【rte_sws_1331】.

3.8.7　Rte_Call

C 言語 I/F	Std_ReturnType Rte_Call_<p>_<o> ([IN|IN/OUT|OUT] <data_1>, ...[IN|IN/OUT|OUT] <data_n>)【rte_sws_1102】	
パラメータ[in]	<data_1>...<data_n>	オペレーション呼出しの引数
パラメータ[in/out]		
パラメータ[out]		
返り値	RTE_E_OK	呼出し処理成功【rte_sws_1104】.
	RTE_E_UNCONNECTED	クライアントポートが未接続【rte_sws_7656】.
	RTE_E_SEG_FAULT	RTE API に渡されるパラメータに対して，〔rte_sws_2752〕，〔rte_sws_2753〕のセグメンテーション違反を検出したときは，オペレーション呼び出しせずに，この値を返す【rte_sws_2755】.
	RTE_E_DEV_DEFECT	A-RTE がデータ伝搬で使用する OS のシステムサービスにおいて，拡張エラーが発生したときはこの値を返す【irte_sws_0022】.
	アプリケーションエラー	同期 C/S 連携では，RTE_E_OK 以外のどの通信基盤エラーも発生しなかった場合のみ，サーバから渡されたアプリケーションエラーを返す【rte_sws_2577】.
同期/非同期	同期	
リエントラント	○	
生成有無の設定	クライアントオペレーションに対し，*同期サーバ呼出しポイント(SyncronousServerCallPoint)*が存在する場合，本 API を生成する【rte_sws_1293】.	
機能		
オペレーション呼出しを行う. ・　<p>はクライアントポートのショートネーム.		

- <o>はクライアントオペレーションのショートネーム.
- <data_1>…<data_n>はオペレーション引数．型は各オペレーション引数の*実装データ型(ImplementationDataType)*，もしくは*実装データ型(ImplementationDataType)*への参照型.

API 呼出しの制限

本 API は対応する*サーバ呼出しポイント(ServerCallPoint)*を含むランナブルのみから使用されなければならない【rte_sws_ext_2685】.

参照渡しの全ての引数のポインタは本 API から復帰するまで有効でなければならない【rte_sws_ext_a_0006】.

機能仕様

本 API は，オペレーション呼出を行う．同期 Rte_Call 呼び出し時に，サーバの処理中に発生した通信基盤エラーとアプリケーションエラーを本 API の返り値として返す【rte_sws_1103】.

C/S 連携の未接続のクライアントポートに対応する Rte_Call は，何も行わず，即座に RTE_E_UNCONNECTED を返す【rte_sws_1334】.

3.8.8　Rte_IrvRead

<table>
<tr><td>C 言語 I/F</td><td colspan="2">本 API の C 言語 I/F は以下の通りである【rte_sws_3560】

<table><tr><td>実装データ型</td><td>C 言語 I/F</td></tr><tr><td>プリミティブデータ型</td><td><return> Rte_IrvRead_<re>_<o> (void)</td></tr><tr><td>上記以外</td><td>void Rte_IrvRead_<re>_<o> (OUT <data>)</td></tr></table></td></tr>
<tr><td>パラメータ[in]</td><td colspan="2">－</td></tr>
<tr><td>パラメータ[in/out]</td><td colspan="2">－</td></tr>
<tr><td>パラメータ[out]</td><td><data></td><td>受信したデータを格納する領域へのポインタ</td></tr>
<tr><td>返り値</td><td><return></td><td>受信したデータ</td></tr>
<tr><td>同期/非同期</td><td colspan="2">同期</td></tr>
<tr><td>リエントラント</td><td colspan="2">○</td></tr>
<tr><td>生成有無の設定</td><td colspan="2">IRV 受信アクセスと IRV 変数データが接続関係にある場合，本 API を生成する【rte_sws_1305】.</td></tr>
<tr><td colspan="3">機能</td></tr>
<tr><td colspan="3">ランナブル間連携の受信データ取得を行う.
・ <re>はランナブルのショートネーム.
・ <o>は IRV 変数データのショートネーム.
・ <data>は取得した受信データの格納先変数への参照であり，データ型は IRV 変数データの実装データ型(ImplementationDataType)への参照型.
・ <return>は取得した受信データであり，データ型は IRV 変数データの実装データ型(ImplementationDataType)への参照型【rte_sws_3562】.</td></tr>
</table>

API 呼出しの制限

<data>が参照渡しである場合，そのポインタは本 API が復帰するまで有効でなければならない【rte_sws_ext_a_0012】.

機能仕様

本 API は, 受信したデータの取得を行う. 本 API は, ECU 間/パーティション間連携を行わないため, エラーを返す必要がない．そのため，プリミティブデータ型については，返り値で受信したデータを返す. プリミティブデータ型以外については, 返り値の型は void で, OUT 引数で受信したデータを返す.

一度もデータを受信していない場合，返り値として IRV 初期値を返す【nrte_sws_0265】.

3.8.9　Rte_IrvWrite

C 言語 I/F	void Rte_IrvWrite_<re>_<o>(IN <data>)【rte_sws_3565】	
パラメータ[in]	<data>	送信するデータ【rte_sws_3567】
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	－【rte_sws_3569】	
同期/非同期	同期	
リエントラント	○	
生成有無の設定	IRV 送信アクセスと IRV 変数データが接続関係にある場合，本 API を生成する【rte_sws_1306】.	
機能		
ランナブル間連携のデータ送信を行う. ・ <re>はランナブルのショートネーム. ・ <o>は IRV 変数データのショートネーム. ・ <data>は送信するデータであり，データ型は IRV 変数データの*実装データ型 (ImplementationDataType)*.		

API 呼出しの制限

<data>が参照渡しである場合，そのポインタは本 API が復帰するまで有効でなければならない【rte_sws_ext_a_0013】.

機能仕様

本 API は，データ送信を行う．本 API は，ECU 間/パーティション間連携を行わないため，エラーを返す必要がない．そのため，返り値の型は void である．

3.8.10 Rte_Enter

C 言語 I/F	void Rte_Enter_<name>(void)【rte_sws_1120】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
同期/非同期	同期
リエントラント	×
生成有無の設定	*排他エリア(ExclusiveArea)に対し，明示的排他エリア入場(canEnterExclusiveArea)*が存在する場合，本 API を生成する【rte_sws_1307】.
機能	
排他エリアに入場する. ・　<name>は排他エリアのショートネーム.	

API 呼出しの制限

本 API は，*明示的排他エリア入場(canEnterExclusiveArea)*の関連付けを持つランナブルからのみ使用されなければならない【rte_sws_ext_7171】.

RTE は同じ排他エリアに対する Rte_Enter のネストされた呼出しをサポートしない.

Rte_Enter/Rte_Exit は，複数の異なる排他エリアに対する入退場が行われ，各排他エリアへの Rte_Exit 呼び出し順序が Rte_Enter 呼び出し順序とは逆順になっている場合のみ，ネストされた呼び出しをサポートする【rte_sws_1122】【rte_sws_ext_7172】.

機能仕様

本 API は排他エリアへの入場を行う.

3.8.11 Rte_Exit

C 言語 I/F	void Rte_Exit_<name>(void)【rte_sws_1123】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
同期/非同期	同期
リエントラント	×
生成有無の設定	*排他エリア(ExclusiveArea)*に対し，*明示的排他エリア入場(canEnterExclusiveArea)*が存在する場合，本 API を生成する【rte_sws_1308】.
機能	
排他エリアから退場する. ・　<name>は排他エリアのショートネーム.	

API 呼出しの制限

本 API は，*明示的排他エリア入場(canEnterExclusiveArea)*の関連付けを持つランナブルからのみ使用されなければならない〔rte_sws_ext_7171〕.

機能仕様

本 API は排他エリアからの退場を行う.

3.9　RTE ライフサイクル API

本章では RTE の開始と終了のための関数(RTE ライフサイクル API)を説明する.

RTE ライフサイクル API は SW-C から使用されず，EcuStateManager から呼ばれる.

3.9.1　Rte_Start

C 言語 I/F	Std_ReturnType　Rte_Start(void)【rte_sws_2569】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	RTE_E_OK	成功【rte_sws_1261】.
同期/非同期	同期	
リエントラント	同一コアからの呼出しに関して×，それ以外は○	
生成有無の設定	常に本 API を生成する【rte_sws_1309】. 設定は不要である.	
機能		
RTE 自身の起動を行う.		

API 呼出しの制限

本 API は，RTE が必要とする以下の BSWM の初期化の後，信頼コンテキストから，EcuStateManager によって呼び出されなければならない【rte_sws_ext_2582】.

- OS
- COM
- メモリサービス

本 API は，ECU 上の SW-C が動作する各コア上で呼び出されなければならない【rte_sws_ext_2714】.

本 API は，SW-C から呼び出してはならない【rte_sws_ext_a_0007】.

本 API は，SCHM が起動した(SchM_Init の実行が終了)後に，呼び出されなければならない【rte_sws_ext_7577】.

機能仕様

本 API は，RTE で使用するシステムと通信のリソースの割当て，および初期化を行う．リソースの割当てに失敗した場合，返り値としてエラーを返す.

本 API は，有限の実行時間で復帰する，つまり無限ループに陥ってはならない【rte_sws_2585】.

本 API は，ライフサイクルヘッダ Rte_Main.h で定義される.

本 API は，関数，もしくはマクロで実装される.

3.9.2　Rte_Stop

C 言語 I/F	Std_ReturnType　Rte_Stop(void)【rte_sws_2570】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	RTE_E_OK	成功【rte_sws_1259】.
同期/非同期	同期	
リエントラント	同一コアからの呼出しに関して×，それ以外は○	
生成有無の設定	常に本 API を生成する【rte_sws_1310】．設定は不要である．	
機能		
RTE 自身を停止する．		

API 呼出しの制限

本 API は，RTE が必要とする以下の BSWM の終了処理の後で，EcuStateManager によって呼び出されなければならない【rte_sws_ext_2583】．

- OS
- COM
- メモリサービス

本 API は，信頼コンテキストから呼び出さなければならず，SW-C から呼び出すべきではない【rte_sws_ext_a_0008】．

機能仕様

本 API は，呼び出されたコア上の RTE を終了するために使用する．

本 API は，有限の実行時間で復帰する，つまり無限ループに陥ってはいけない【rte_sws_2584】．

本 API は，ライフサイクルヘッダ Rte_Main.h で定義される．

本 API は，関数，もしくはマクロで実装される．

3.9.3　Rte_PartitionTerminated

C 言語 I/F	void Rte_PartitionTerminated_<PID>(void)【rte_sws_7330】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
同期/非同期	同期
リエントラント	×
生成有無の設定	パーティション毎に，本 API を生成する【rte_sws_7331】.
機能	
パーティションが停止され，そのパーティションとの通信が無効になることを RTE に通知する. ・　<PID>はパーティションのショートネーム.	

API 呼出しの制限

本 API は，OS プロテクションフックによって 1 度だけ呼び出されなければならない【rte_sws_ext_7332】．本 RTE では，異なるコア間の N:1 連携において OS スピンロックを使った排他制御によるデータ一貫性保証を行っており，本 API が呼び出された場合にも排他制御する必要がある．そのため，OS プロテクションフックのコンテキストで OS スピンロックを獲得できなければならない【nrte_sws_ext_0004】.

機能仕様

本 API は，指定パーティションが停止されることを RTE に通知する.

パーティションが停止状態の場合，本 API は何も行わず復帰する【rte_sws_7335】.

本 API は，ライフサイクルヘッダ Rte_Main.h で定義される.

本 API 内の処理は，OS プロテクションフックのコンテキストで許可されている API のみを使用する【rte_sws_7334】.

本 API は，関数あるいはマクロで実装される.

3.9.4　Rte_PartitionRestarting

C 言語 I/F	void Rte_PartitionRestarting_<PID>(void)【rte_sws_7620】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
同期/非同期	同期
リエントラント	×
存在条件	再起動可能なパーティション(つまり *パーティション再起動可能フラグ(PartitionCanBeRestarted)*が再起動可能のパーティション)に対し，本 API を生成する【rte_sws_7619】.
機能	
パーティションが再起動され，そのパーティションとの通信が無効になることを RTE に通知する. ・　<PID>はパーティションのショートネーム.	

API 呼出しの制限

本 API は，OS プロテクションフックによって 1 度だけ呼び出されなければならない【rte_sws_ext_7618】．本 RTE では，異なるコア間の N:1 連携において OS スピンロックを使った排他制御によるデータ一貫性保証を行っており，本 API が呼び出された場合にも排他制御する必要がある．そのため，OS プロテクションフックのコンテキストで OS スピンロックを獲得できなければならない〔nrte_sws_ext_0004〕.

機能仕様

本 API は，指定パーティションが再起動されることを RTE に通知する．Rte_PatitionTerminated と同様にそのパーティションとの通信が無効になることを示すが，そのパーティションがいずれ再起動する点が異なる.

パーティションが停止状態，もしくは再起動中状態の場合，本 API は何も行わず復帰する【rte_sws_7622】.

本 API は，ライフサイクルヘッダ Rte_Main.h で定義される.

本 API 内の処理は，OS プロテクションフックのコンテキストで許可されている API のみを使用する【rte_sws_7617】.

本 API は，関数あるいはマクロで実装される.

3.9.5　Rte_RestartPartition

C 言語 I/F	Std_ReturnType　Rte_RestartPartition_<PID>(void) 【rte_sws_7188】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	RTE_E_OK	成功【rte_sws_7341】.
同期/非同期	同期	
リエントラント	×	
生成有無の設定	再起動可能なパーティション(つまり *パーティション再起動可能フラグ(PartitionCanBeRestarted)*が再起動可能のパーティション)に対し，本 API を生成する【rte_sws_7336】.	
機能		
パーティションに割り当てた RTE のリソースを初期化する. ・　<PID>はパーティションのショートネーム.		

API 呼出しの制限

本 API は，指定パーティションの OS リスタートタスクのコンテキストのみから呼び出されなければならない【rte_sws_ext_7337】.

機能仕様

本 API は，パーティションに割り当てた RTE のリソースを初期化する．リソースの割当てに失敗した場合，返り値としてエラーを返す.

本 API は，有限の実行時間で復帰する，つまり無限ループに陥ってはならない【rte_sws_7338】.

本 API は，以下に示すパーティションの RTE 環境を初期化し，このパーティションとの通信を再起動しなければならない【rte_sws_7339】.

・　データ要素

パーティションが動作状態の場合，本 API は，何も行わず復帰する【rte_sws_7340】.

本 API は，ライフサイクルヘッダ Rte_Main.h で定義される.

本 API は，関数あるいはマクロで実装される.

3.10 RTE コールバック

本節では，RTEGEN が生成するコールバック関数について記載する．

このコールバック関数は，通信サービスなどの他 SW-C によって呼び出されるため，明確な名前と意味を持つ必要がある．

RTEGEN は，COM から RTE に対して COM イベントを通知するためのコールバック関数を提供する【rte_sws_3530】．RTEGEN が生成するコールバック実装は，有限の時間で復帰し，ブロックを行わない【rte_sws_1165】．

本 RTEGEN がコールバック関数を生成する条件は，実装定義とする【nrte_sws_0043】．RTEGEN は，Rte_Cbk ヘッダに RTE が必要とする全てのコールバック関数の宣言を生成する．本 RTE は，COM のコンフィギュレーションで各コールバック関数の宣言に対応するコールバック設定が行われることを期待する．

3.10.1 COM コールバック

C 言語 I/F	void <CallbackRoutineName> (void)【rte_sws_3000】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
機能	
ECU 間での通信結果を通知するために COM が呼び出すコールバック関数の実装． ・　<CallbackRoutineName>はコールバック関数名	

機能仕様

<CallbackRoutineName>のプロトタイプは COM が提供する．

COM から実際に呼び出されるコールバック内容(例：データを受信した)は，COM のコンフィギュレーションによって決定される．

3.10.1.1 期待する COM コンフィギュレーション

以下に，RTE の提供する各 COM コールバックに対し，期待する COM コンフィギュレーション内容を定義する．

表 3-1 RTE COM コールバック一覧

コールバック関数名	説明	コンフィギュレーション内容	仕様番号
Rte_COMCbk_<sn>	プリミティブ型のデータやイベントの受信準備ができている際に呼び出されることを期待する.	*COM シグナル(ComSignal)* の *COM 受信通知 (ComNotification)*に本コールバックを設定する.	【rte_sws_3001】
Rte_COMCbkInv_<sn>	"Inv"はシグナル無効化を示す. COM がシグナルを受信したが無効と判断した際に呼び出されることを期待する.	*COM シグナル(ComSignal)* の *COM 無効化通知 (ComInvalidNotification)*に本コールバックを設定する.	【rte_sws_2612】
Rte_COMCbkRxTOut_<sn>	"RxTOut"は受信シグナルのタイムアウトを示す. プリミティブ型データとイベントのシグナルの最後の受信後, タイムアウト時間経過した(データ要素が古い)際に呼び出されることを期待する.	*COM シグナル(ComSignal)* の *COM タイムアウト通知 (ComTimeoutNotification)*に本コールバックを設定する.	【rte_sws_2610】
Rte_COMCbkTAck_<sn>	"TAck"は送信 ACK を示す. プリミティブ型データとイベントのシグナルが, COM によって PDUR に伝達された際に呼び出されることを期待する.	*COM シグナル(ComSignal)* の *COM 通知 (ComNotification)*に本コールバックを設定する.	【rte_sws_3002】
Rte_COMCbkTErr_<sn>	"TErr"は送信エラーを示す. プリミティブ型データとイベントのシグナルが, COM による PDUR への伝達においてエラーが検出された際に呼び出されることを期待する.	*COM シグナル(ComSignal)* の *COM エラー通知 (ComErrorNotification)*に本コールバックを設定する.	【rte_sws_3775】
Rte_COMCbkTxTOut_<sn>	"TxTOut"は送信失敗によるタイムアウトを示す. 複合データ型データとイベントのシグナルが, 送信 ACK 要求のタイムアウト時間経過した際に呼び出されることを期待する.	*COM シグナル(ComSignal)* の *COM タイムアウト通知 (ComTimeoutNotification)*に本コールバックを設定する.	【rte_sws_5084】
Rte_COMCbk	複合データ型のデータやイベン	*COM シグナルグループ*	【rte_sws_3004】

_<sg>	トの受信準備ができている際に呼び出されることを期待する.	*(ComSignalGroup)の COM 通知(ComNotification)*に本コールバックを設定する.	
Rte_COMCbkRxTOut_<sg>	"RxTOut"は受信シグナルのタイムアウトを示す. 複合データ型データのシグナルの最後の受信後，タイムアウト時間経過した(データ要素が古い)際に呼び出されることを期待する.	*COM シグナルグループ(ComSignalGroup)の COM タイムアウト通知(ComTimeoutNotification)*に本コールバックを設定する.	【rte_sws_2611】
Rte_COMCbkTAck_<sg>	"TAck"は送信 ACK を示す. 複合データ型データとイベントのシグナルが，COM によって PDUR に伝達された際に呼び出されることを期待する.	*COM シグナルグループ(ComSignalGroup)の COM 通知(ComNotification)*に本コールバックを設定する.	【rte_sws_3005】
Rte_COMCbkTErr_<sg>	"TErr"は送信エラーを示す. 複合データ型データとイベントのシグナルが，COM による PDUR への伝達においてエラーが検出された際に呼び出されることを期待する.	*COM シグナルグループ(ComSignalGroup)の COM エラー通知(ComErrorNotification)*に本コールバックを設定する.	【rte_sws_3776】
Rte_COMCbkTxTOut_<sg>	"TxTOut"は送信失敗によるタイムアウトを示す. 複合データ型データとイベントのシグナルが, 送信 ACK 要求のタイムアウト時間経過した際に呼び出されることを期待する.	*COM シグナルグループ(ComSignalGroup)の COM タイムアウト通知(ComTimeoutNotification)*に本コールバックを設定する.	【rte_sws_5085】

<sn>：*COM シグナル(ComSignal)*のショートネーム.

<sg>：*COM シグナルグループ(ComSignalGroup)*のショートネーム.

3.11 SCHM API

3.11.1 SchM_Enter

C 言語 I/F	void SchM_Enter_<bsnp>_<name>(void)【rte_sws_7250】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
同期/非同期	同期
リエントラント	×
生成有無の設定	*排他エリア(ExclusiveArea)に対し，明示的排他エリア入場(canEnterExclusiveArea)*が存在する場合，本 API を生成する【rte_sws_7251】.
機能	
排他エリアに入場する. ・ <bsnp>は BSWM ディスクリプションのショートネーム〔rte_sws_7593〕. ・ <name>は排他エリアのショートネーム.	

API 呼出しの制限

本 API は，*明示的排他エリア入場(canEnterExclusiveArea)*の関連付けを持つ BSWM エンティティからのみ使用されなければならない【rte_sws_ext_7285】.

SCHM は同じ排他エリアに対する SchM_Enter のネストされた呼出しをサポートしない.

SchM_Enter/SchM_Exit は，複数の異なる排他エリアに対する入退場が行われ，各排他エリアへの SchM_Exit 呼び出し順序が SchM_Enter 呼び出し順序とは逆順になっている場合のみ，ネストされた呼び出しをサポートする【rte_sws_7252】【rte_sws_ext_7529】.

機能仕様

本 API は排他エリアへの入場を行う.

SchM_Enter/SchM_Exit は，SCHM が起動していなくても，OS の初期化後であれば呼び出し可能でなければならない【rte_sws_7578】.

SchM_Enter/SchM_Exit は，以下のコンテキストで呼び出し可能でなければならない【rte_sws_7579】.

- OS タスク
- C1ISR
- C2ISR

ただし，呼び出しコンテキストは排他エリアの実現メカニズムによって制約される．たとえば，実現メカニズムが「全割込みのブロック」である場合は，全てのコンテキストで呼び出し可能であるが，「OS

リソースの獲得」の場合は，呼び出しコンテキストは OS タスクと C2ISR のみが有効である．

AUTOSAR 仕様との違い

AUTOSAR 仕様では，SchM_Enter に，BSWM のベンダ *ID(vendorId)*とベンダ *API 接中辞 (vendorApiInfix)*を接頭辞に含めている【rte_sws_7250】.

しかし, A-RTEGEN では, SCHM のマルチインスタンスはサポートしないため, ベンダ *ID(vendorId)* とベンダ *API 接中辞(vendorApiInfix)*は使用しない〔irte_sws_0009〕.

3.11.2 SchM_Exit

C 言語 I/F	void SchM_Exit_<bsnp>_<name>(void)【rte_sws_7253】
パラメータ[in]	－
パラメータ[in/out]	－
パラメータ[out]	－
返り値	－
同期/非同期	同期
リエントラント	×
生成有無の設定	*排他エリア(ExclusiveArea)*に対し，*明示的排他エリア入場(canEnterExclusiveArea)*が存在する場合，本 API を生成する【rte_sws_7254】.
機能	
排他エリアから退場する. ・ <bsnp>は BSWM ディスクリプションのショートネーム〔rte_sws_7593〕. ・ <name>は排他エリアのショートネーム.	

API 呼出しの制限

本 API は，*明示的排他エリア入場(canEnterExclusiveArea)*の関連付けを持つ BSWM エンティティからのみ使用されなければならない【rte_sws_ext_7189】.

機能仕様

本 API は排他エリアからの退場を行う.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，SchM_Exit に，BSWM の*ベンダ ID(vendorId)*と*ベンダ API 接中辞(vendorApiInfix)*を接頭辞に含めている【rte_sws_7253】.

しかし, A-RTEGEN では, SCHM のマルチインスタンスはサポートしないため, *ベンダ ID(vendorId)*と*ベンダ API 接中辞(vendorApiInfix)*は使用しない〔irte_sws_0009〕.

3.11.3 SchM_Switch

C 言語 I/F	Std_ReturnType SchM_Switch_<bsnp>_<name>(IN <mode>) 【rte_sws_7255】	
パラメータ[in]	<mode>	モード切替通知で要求するモードであり，データ型はモード宣言グループに紐付けられた実装データ型〔nrte_sws_0347〕.
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	SCHM_E_OK	要求された<mode>が正常に渡された【rte_sws_7258】.
	SCHM_E_LIMIT	キューが一杯になり，要求された<mode>が捨てられた【rte_sws_7259】.
	SCHM_E_DEV_DEFECT	A-SCHM が OS のシステムサービスにおいて，拡張エラーが発生したときはこの値を返す【irte_sws_0026】.
同期/非同期	非同期	
リエントラント	○	
生成有無の設定	モードマシンインスタンスのモード切替のアクセス設定が*提供モードグループ(providedModeGroup)*にある場合，本 API を生成する【rte_sws_7256】.	
機能		
モード切替通知を行う. ・ <bsnp>は BSWM ディスクリプションのショートネーム〔rte_sws_7593〕. ・ <name>はモード宣言グループプロトタイプ(提供モードグループ)のショートネーム.		

API 呼出しの制限

本 API は対応する*モードグループ管理(managedModeGroup)*を含む BSWM エンティティのみから使用されなければならない【rte_sws_ext_7257】.

機能仕様

本 API はモード切替通知を行う．詳細は 2.11.4.1 節を参照.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，SchM_Switch に，BSWM の*ベンダ ID(vendorId)*と*ベンダ API 接中辞(vendorApiInfix)*を接頭辞に含めている【rte_sws_7255】.

しかし, A-RTEGEN では, SCHM のマルチインスタンスはサポートしないため, *ベンダ ID(vendorId)* と *ベンダ API 接中辞(vendorApiInfix)*は使用しない〔irte_sws_0009〕.

AUTOSAR 仕様(R4.0.3)では，SchM_Switch の引数の型をモードのデータ型と定義しているが，モードのデータ型宣言〔rte_sws_7292〕は非推奨としている．この不整合を解消するため，AUTOSAR 仕様(R4.2.1)に合わせ，引数の型には，モード宣言グループ *(ModeDeclarationGroup)*に紐付けられた実装データ型*(ImplementationDataType)*を使用する【nrte_sws_0347】.

3.11.4 SchM_Mode

C 言語 I/F	<return> SchM_Mode_<bsnp>_<name>(void)【rte_sws_7260】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	RTE_TRANSITION_ <ModeDeclarationGroup>	モード遷移中の場合，モード遷移中のステータスを返す【rte_sws_7262】.
	RTE_MODE_ <ModeDeclarationGroup>_ <ModeDeclaration>	モード遷移中でない場合，現在のモードを返す【rte_sws_7262】.
同期/非同期	同期	
リエントラント	○	
生成有無の設定	モードマシンインスタンスのモード参照のアクセス設定が*提供モードグループ(providedModeGroup)*または*要求モードグループ(requiredModeGroup)*にある場合，本 API を生成する【rte_sws_7261】.	

機能

モード取得を行う.

- <bsnp>は BSWM ディスクリプションのショートネーム〔rte_sws_7593〕.
- <name>はモード宣言グループプロトタイプ(要求モードグループ)のショートネーム.
- <ModeDeclarationGroup>はモード宣言グループのショートネーム.
- <ModeDeclaration>はモード宣言のショートネーム.
- <return>のデータ型はモード宣言グループに紐付けられた実装データ型〔nrte_sws_0348〕.

API 呼出しの制限

本 API は対応する *モードグループ管理(managedModeGroup)*または*モードグループ参照(accessedModeGroup)*を含む BSWM エンティティのみから使用されなければならない【rte_sws_ext_7587】.

機能仕様

本 API はモード取得を行う．詳細は 2.11.4.3 節を参照.

AUTOSAR 仕様との違い

AUTOSAR 仕様では，SchM_Mode に，BSWM のベンダ *ID(vendorId)*と ベンダ *API* 接中辞 *(vendorApiInfix)*を接頭辞に含めている【rte_sws_7260】.

しかし, A-RTEGEN では, SCHM のマルチインスタンスはサポートしないため, ベンダ *ID(vendorId)* と ベンダ *API* 接中辞*(vendorApiInfix)*は使用しない〔irte_sws_0009〕.

AUTOSAR 仕様(R4.0.3)では，SchM_Mode の返り値の型をモードのデータ型と定義しているが，モードのデータ型宣言〔rte_sws_7292〕は非推奨としている．この不整合を解消するため，AUTOSAR 仕様(R4.2.1)に合わせ，返り値の型には，モード宣言グループ *(ModeDeclarationGroup)*に紐付けられた実装データ型*(ImplementationDataType)*を使用する【nrte_sws_0348】.

3.12 SCHM ライフサイクル API

本章では SCHM の開始と終了のための関数(SCHM ライフサイクル API)を説明する．

SCHM ライフサイクル API は BSWM から使用されず，EcuStateManager から呼ばれる．

3.12.1 SchM_Init

C 言語 I/F	void SchM_Init(void)【rte_sws_7270】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	－	－
同期/非同期	同期	
リエントラント	同一コアからの呼出しに関して×，それ以外は○	
生成有無の設定	常に本 API を生成する【rte_sws_7271】．設定は不要である．	
機能		
SCHM 自身の起動を行う．		

API 呼出しの制限

本 API は，SCHM が必要とする以下の BSWM の初期化の後，信頼コンテキストから，EcuStateManager によって呼び出されなければならない．

・ OS

また，本 API の呼び出し回数は，各コアで 1 回のみでなければならない【rte_sws_ext_7272】．

機能仕様

本 API は，呼び出されたコア上で動作する SCHM のリソースの割当て，および初期化を行う．本 API 呼び出し後，BSW スケジューラブルのスケジューリングが有効となる．

本 API は，有限の実行時間で復帰する，つまり無限ループに陥ってはならない【rte_sws_7273】．

本 API は，ライフサイクルヘッダ Rte_Main.h で定義される．

本 API は，関数，もしくはマクロで実装される．

3.12.2 SchM_Deinit

C 言語 I/F	void SchM_Deinit(void)【rte_sws_7274】	
パラメータ[in]	－	
パラメータ[in/out]	－	
パラメータ[out]	－	
返り値	－	－
同期/非同期	同期	
リエントラント	同一コアからの呼出しに関して×，それ以外は○	
生成有無の設定	常に本 API を生成する【rte_sws_7275】．設定は不要である．	
機能		
SCHM 自身を停止する．		

API 呼出しの制限

本 API は，SCHM が必要とする以下の BSWM の終了処理の前に，EcuStateManager によって呼び出されなければならない．

・ OS

また，本 API の呼び出し回数は，各コアで 1 回のみでなければならない【rte_sws_ext_7276】．

本 API は，Rte_Stop 呼び出し終了後にのみ呼び出されなければならない【rte_sws_ext_7576】．

機能仕様

本 API は，呼び出されたコア上の SCHM を終了するために使用する．

本 API は，有限の実行時間で復帰する，つまり無限ループに陥ってはいけない【rte_sws_7277】．

本 API は，ライフサイクルヘッダ Rte_Main.h で定義される．

本 API は，関数，もしくはマクロで実装される．

3.13 依存インタフェース

RTE は, AUTOSAR コンポーネントに RTE 機能を提供するために, AUTOSAR OS と AUTOSAR COM のみ利用できる【rte_sws_2250】.

3.13.1 想定する OS インタフェース

本 RTE/SCHM は以下の OS システムサービスを使用する.

OS システムサービス	概要
ActivateTask	指定 OS タスクの状態を suspend から ready にする.
TerminateTask	自分の OS タスクの状態を running から suspend にする.
SuspendAllInterrupts	全割込み禁止.
ResumeAllInterrupts	全割込み許可.
SuspendOSInterrupts	OS 割込み禁止.
ResumeOSInterrupts	OS 割込み許可.
GetResource	OS リソースの獲得.
ReleaseResource	OS リソースの解放.
SetEvent	イベントのセット.
ClearEvent	イベントのクリア.
GetEvent	イベントの状態参照.
WaitEvent	イベント待ち.
CallTrustedFunction	信頼関数を, プロセッサコアを特権モードに変更して呼び出す.
GetTaskID	現在実行中のタスクの ID 情報を取得する.
CheckISRMemoryAccess	指定メモリ領域に対する ISR のアクセス権の取得.
CheckTaskMemoryAccess	指定メモリ領域に対する OS タスクのアクセス権の取得.
GetSpinlock	指定スピンロックの獲得.
ReleaseSpinlock	指定スピンロックの解放.

本 RTE/SCHM は，引数に ID を必要とする OS システムサービスを呼びだす場合，OS オブジェクトの ID として OS オブジェクトのショートネームを使用して，OS システムサービスを呼びだす【nrte_sws_0169】.

3.13.2 想定する IOC インタフェース

本 RTE は以下の IOC API を使用する.

IOC システムサービス	概要
IocWrite	IOC(キューなし通信，単一通信)の送信処理を行う.
IocRead	IOC(キューなし通信，単一通信)の受信処理を行う.
IocSend	IOC(キューあり通信，単一通信)の送信処理を行う.
IocReceive	IOC(キューあり通信，単一通信)の受信処理を行う.
IocSendGroup	IOC(キューあり通信，グループ通信)の送信処理を行う.
IocReceiveGroup	IOC(キューあり通信，グループ通信)の受信処理を行う.
IocEmptyQueue	IOC のキューの初期化を行う.

3.13.3 依存する COM インタフェース

本 RTE は以下の COM API を使用する.

COM API	概要
Com_SendSignal	COM を使用してプリミティブ型のデータを送信する. **ComShadowSignal API 使用有無 (RteUseComShadowSignalApi)**が false の場合，複合データ型の実装データ型要素を送信する.
Com_ReceiveSignal	COM を使用してプリミティブ型のデータを受信する. **ComShadowSignal API 使用有無 (RteUseComShadowSignalApi)**が false の場合，複合データ型の実装データ型要素を受信する.
Com_InvalidateSignal	COM を使用してプリミティブ型のデータを無効化する.
Com_UpdateShadowSignal	**ComShadowSignal API 使用有無 (RteUseComShadowSignalApi)**が true の場合，複合データ型の実装データ型要素を送信する.
Com_ReceiveShadowSignal	**ComShadowSignal API 使用有無 (RteUseComShadowSignalApi)**が true の場合，複合データ型の実装データ型要素を受信する.
Com_SendSignalGroup	COM を使用して複合データ型のデータを送信する.
Com_ReceiveSignalGroup	COM を使用して複合データ型のデータを受信する.

本 RTE は，COM シグナルの ID として，関連文書「Specification of ECU Configuration」の〔ecuc_sws_2108〕で規定されるシンボルを使用して，COM API を呼びだす【nrte_sws_0168】.

非信頼パーティションから COM API を呼び出す場合，RTE は，OS 信頼関数経由で COM API を呼び出す【rte_sws_2761】．本 RTE が使用する COM API はリエントラントでなければならない【nrte_sws_ext_0005】.

3.14 コンテナ

3.14.1 Rte

コンテナ名	/AUTOSAR/EcucDefs/Rte【rte_sws_a_0050】
概要	RTE トップコンテナ
多重度	0..1
パラメータ	－
サブコンテナ	RteBswGeneral
	RteBswModuleInstance
	RteGeneration
	RteImplicitCommunication(サポート対象外)
	RteInitializationBehavior
	RteOsInteraction
	RtePostBuildVariantConfiguration(サポート対象外)
	RteSwComponentInstance
	RteSwComponentType(サポート対象外)
制限事項	－

3.14.2 RteBswGeneral

コンテナ名	/AUTOSAR/EcucDefs/Rte/ RteBswGeneral【rte_sws_9061_Conf】
概要	SCHM の一般的なコンフィギュレーションパラメータを保持する
多重度	1
パラメータ	RteSchMVersionInfoApi(サポート対象外)
	RteUseComShadowSignalApi
サブコンテナ	－
制限事項	－

RteUseComShadowSignalApi

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswGeneral RteUseComShadowSignalApi【rte_sws_9107_Conf】
概要	ComShadowSignal API(※)の使用有無 (※)Com_UpdateShadowSignal, Com_InvalidateShadowSignal, Com_ReceiveShadowSignal
型	ブール型
値の範囲	true：ComShadowSignal API および ComSignal API を使用 false：ComSignal API(※)のみ使用 (※)Com_SendSignal，Com_InvalidateSignal, Com_ReceiveSignal
多重度	1
制限事項	－

3.14.3 RteBswModuleInstance

コンテナ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance 【rte_sws_9002_Conf】
概要	SCHM 生成に関するパラメータを保持する
多重度	0..*
パラメータ	RteBswImplementationRef
	RteBswModuleConfigurationRef (サポート対象外)
	RteBswEventToTaskMapping
	RteBswExclusiveAreaImpl
	RteBswExternalTriggerConfig (サポート対象外)
	RteBswInternalTriggerConfig (サポート対象外)
	RteBswRequiredModeGroupConnection
	RteBswRequiredTriggerConnection (サポート対象外)
サブコンテナ	－
制限事項	－

RteBswImplementationRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance/ RteBswImplementationRef【rte_sws_9066_Conf】
概要	*BSW 実装(BswImplementaion)*
型	*BSW 実装(BswImplementaion)*への参照
値の範囲	－
多重度	1
制限事項	－

3.14.4 RteBswEventToTaskMapping

コンテナ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance/ RteBswEventToTaskMapping【rte_sws_9065_Conf】
概要	BSW イベント(および SCHM イベントから起動する BSW スケジューラブル)の OS タスクへのマッピング
多重度	0..*
パラメータ	RteBswActivationOffset
	RteBswEventRef
	RteBswImmediateRestart(サポート対象外)
	RteBswMappedToTaskRef
	RteBswPositionInTask
	RteBswUsedOsAlarmRef
	RteBswUsedOsEventRef
	RteBswUsedOsSchTblExpiryPointRef(サポート対象外)
	RteOsSchedulePoint (サポート対象外)
サブコンテナ	－
制限事項	－

RteBswActivationOffset

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance / RteBswEventToTaskMapping/ RteBswActivationOffset【rte_sws_9063_Conf】
概要	BSW イベントの起動オフセット(単位：秒)
型	浮動小数点型
値の範囲	0 ..無限大
多重度	0..1
制限事項	－

RteBswEventRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance / RteBswEventToTaskMapping/ RteBswEventRef【rte_sws_9064_Conf】
概要	コンフィギュレーション対象の BSW イベントへの参照
型	**BSW イベント(BswEvent)**への参照
値の範囲	－
多重度	1
制限事項	－

RteBswMappedToTaskRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance / RteBswEventToTaskMapping/ RteBswMappedToTaskRef【rte_sws_9067_Conf】
概要	BSW イベントのマッピング先の OS タスク
型	/AUTOSAR/EcucDefs/Os/OsTask への参照
値の範囲	－
多重度	0..1
制限事項	－

RteBswPositionInTask

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance / RteBswEventToTaskMapping/ RteBswPositionInTask【rte_sws_9068_Conf】
概要	BSW スケジューラブルの実行順番
型	整数型
値の範囲	0..65535
多重度	0..1
制限事項	－

RteBswUsedOsAlarmRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance / RteBswEventToTaskMapping/ RteBswUsedOsAlarmRef【rte_sws_9069_Conf】
概要	BSW イベントのマッピング先 OS タスクを起動する OS アラーム
型	/AUTOSAR/EcucDefs/Os/OsAlarm への参照
値の範囲	－
多重度	0..1
制限事項	－

RteBswUsedOsEventRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance / RteBswEventToTaskMapping/ RteUsedOsEventRef【rte_sws_9070_Conf】
概要	BSW イベントのマッピング先 OS イベント
型	/AUTOSAR/EcucDefs/Os/OsEvent への参照
値の範囲	－
多重度	0..1
制限事項	－

3.14.5 RteBswExclusiveAreaImpl

コンテナ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance/ RteBswExclusiveAreaImpl【rte_sws_9072_Conf】
概要	排他エリアの実現メカニズム
多重度	0..*
パラメータ	RteBswExclusiveAreaOsResourceRef
	RteBswExclusiveAreaRef
	RteExclusiveAreaImplMechanism
	RteBswExclusiveAreaOsSpinlockRef
サブコンテナ	－
制限事項	－

RteBswExclusiveAreaOsResourceRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance/ RteBswExclusiveAreaImpl/ RteBswExclusiveAreaOsResourceRef【rte_sws_9073_Conf】
概要	排他エリアの実現のために使用する OS リソース
型	/AUTOSAR/EcucDefs/Os/OsResource への参照
値の範囲	－
多重度	0..1
制限事項	－

RteBswExclusiveAreaRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance / RteBswExclusiveAreaImpl / RteBswExclusiveAreaRef【rte_sws_9074_Conf】
概要	コンフィギュレーション対象の排他エリア
型	*排他エリア(ExclusiveArea)*への参照
値の範囲	－
多重度	1
制限事項	－

RteExclusiveAreaImplMechanism

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance / RteBswExclusiveAreaImpl / RteExclusiveAreaImplMechanism【rte_sws_9029_Conf】
概要	排他エリアの実現メカニズム種別
型	列挙型
値の範囲	ALL_INTERRUPT_BLOCKING：全割込みの禁止 COOPERATIVE_RUNNABLE_PLACEMENT：協調ランナブル配置 OS_INTERRUPT_BLOCKING：OS 割込みの禁止 OS_RESOURCE：OS リソースの獲得 OS_SPINLOCK：OS スピンロックの獲得 NONE：排他なし
多重度	1
制限事項	本 RTE では，COOPERATIVE_RUNNABLE_PLACEMENT をサポートしない【nrte_sws_0048】.

RteBswExclusiveAreaOsSpinlockRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance / RteBswExclusiveAreaImpl / RteBswExclusiveAreaOsSpinlockRef【nrte_sws_0066_Conf】
概要	排他エリアの実現のために使用する OS リソース
型	/AUTOSAR/EcucDefs/Os/OsResource への参照
値の範囲	－
多重度	0..1
制限事項	－

3.14.6 RteBswRequiredModeGroupConnection

コンテナ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance/ RteBswRequiredModeGroupConnection【rte_sws_9081_Conf】
概要	要求モードグループと提供モードグループの接続
多重度	0..*
パラメータ	RteBswProvidedModeGroupRef
	RteBswProvidedModeGrpModInstRef
	RteBswRequiredModeGroupRef
サブコンテナ	－
制限事項	－

RteBswProvidedModeGroupRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance/ RteBswRequiredModeGroupConnection/ RteBswProvidedModeGroupRef【rte_sws_9079_Conf】
概要	提供モードグループ
型	**モード宣言グループプロトタイプ (ModeDeclarationGroupPrototype)**への参照
値の範囲	－
多重度	1
制限事項	－

RteBswProvidedModeGrpModInstRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance/ RteBswRequiredModeGroupConnection/ RteBswProvidedModeGrpModInstRef【rte_sws_9080_Conf】
概要	RteBswModuleInstance への参照
型	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance への参照
値の範囲	－
多重度	1
制限事項	－

RteBswRequiredModeGroupRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/ RteBswModuleInstance/ RteBswRequiredModeGroupConnection/ RteBswRequiredModeGroupRef【rte_sws_9082_Conf】
概要	要求モードグループ
型	**モード宣言グループプロトタイプ (ModeDeclarationGroupPrototype)**への参照
値の範囲	－
多重度	1
制限事項	－

3.14.7 RteGeneration

コンテナ名	/AUTOSAR/EcucDefs/Rte/RteGeneration【rte_sws_9009_Conf】
概要	RTE 生成に関するパラメータを保持する
多重度	1
パラメータ	RteCalibrationSupport(サポート対象外)
	RteCodeVendorId(サポート対象外)
	RteDevErrorDetect(サポート対象外)
	RteDevErrorDetectunInit(サポート対象外)
	RteGenerationMode
	RteIocInteractionReturnValue
	RteMeasurementSupport(サポート対象外)
	RteOptimizationMode
	RteToolChainSignificantCharacters(サポート対象外)
	RteValueRangeCheckEnabled(サポート対象外)
	RteVfbTraceClientPrefix(サポート対象外)
	RteVfbTraceEnabled(サポート対象外)
	RteVfbTraceFunction(サポート対象外)
サブコンテナ	－
制限事項	－

RteGenerationMode

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteGeneration/ RteGenerationMode【rte_sws_9010_Conf】
概要	RTE の生成モード
型	列挙型
値の範囲	COMPATIBILITY_MODE：互換モード(デフォルト) VENDOR_MODE：ベンダモード
多重度	1
制限事項	－

RteIocInteractionReturnValue

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteGeneration/ RteIocInteractionReturnValue【rte_sws_9094_Conf】
概要	RTE API の返り値設定
型	列挙型
値の範囲	RTE_COM：COM 準拠 RTE_IOC：IOC 準拠（デフォルト）
多重度	1
制限事項	常に RTE_IOC として扱う

RteOptimizationMode

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteGeneration/ RteOptimizationMode【rte_sws_9012_Conf】
概要	RTE の最適化モード
型	列挙型
値の範囲	MEMORY：メモリ消費量の最適化優先 RUNTIME：実行時間の最適化優先(デフォルト)
多重度	1
制限事項	－

3.14.8 RteInitializationBehavior

コンテナ名	/AUTOSAR/EcucDefs/Rte/RteInitializationBehavior 【rte_sws_9087_Conf】
概要	RTE の内部変数の初期化戦略のコンフィギュレーション
多重度	1..*
パラメータ	RteInitializationStrategy
	RteSectionInitializationPolicy
サブコンテナ	－
制限事項	－

RteInitializationStrategy

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteInitializationBehavior/ RteInitializationStrategy【rte_sws_9089_Conf】
概要	RTE の初期化戦略種別
型	列挙型
値の範囲	RTE_INITIALIZATION_STRATEGY_AT_DATA_DECLARATION：変数の宣言時に初期化. RTE_INITIALIZATION_STRATEGY_AT_ DATA_DECLARATION_AND_PARTITION_RESTART：変数の宣言時，および Rte_RestartPartition の呼び出し時に初期化. RTE_INITIALIZATION_STRATEGY_AT_ RTE_START_AND_PARTITION_RESTART：Rte_Start，および Rte_RestartPartition の呼び出し時に初期化. RTE_INITIALIZATION_STRATEGY_NONE：初期化しない. RTE_INITIALIZATION_STRATEGY_AT_RTE_START：Rte_Start の呼出し時に初期化〔nrte_sws_0056〕.
多重度	1
制限事項	－

RteSectionInitializationPolicy

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteInitializationBehavior/ RteSectionInitializationPolicy【rte_sws_9088_Conf】
概要	RTE の初期化戦略の適用対象のセクション初期化ポリシ. *ソフトウェアアドレッシング方式(SwAddrMethod)のセクション初期化ポリシ(sectionInitializationPolicy)*に指定した値を指定する.
型	文字列型
値の範囲	－
多重度	1..*
制限事項	－

3.14.9 RteOsInteraction

コンテナ名	/AUTOSAR/EcucDefs/Rte/RteOsInteraction 【rte_sws_a_0002_Conf】
概要	RTE-OS 間の連携のコンフィギュレーション
多重度	1..*
パラメータ	－
サブコンテナ	RteUsedOsActivation
	RteModeToScheduleTableMapping(サポート対象外)
制限事項	－

3.14.10RteUsedOsActivation

コンテナ名	/AUTOSAR/EcucDefs/Rte/RteOsInteraction/RteUsedOsActivation 【rte_sws_9060_Conf】
概要	ランナブルの起動のために使用する OS オブジェクト，および OS オブジェクトに期待する設定
多重度	0..*
パラメータ	RteActivationOsAlarmRef
	RteActivationOsSchTblRef(サポート対象外)
	RteActivationOsTaskRef(サポート対象外)
	RteExpectedActivationOffset
	RteExpectedTickDuration
サブコンテナ	－
制限事項	－

RteActivationOsAlarmRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteOsInteraction/RteUsedOsActivation/RteActivationOsAlarmRef【rte_sws_9045_Conf】
概要	ランナブルの起動のために使用する OS アラーム
型	/AUTOSAR/EcucDefs/Os/OsAlarm への参照
値の範囲	－
多重度	0..1
制限事項	－

RteExpectedActivationOffset

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteOsInteraction/RteUsedOsActivation/RteExpectedActivationOffset【rte_sws_9048_Conf】
概要	OS タスク，OS アラーム，もしくは OS スケジュールテーブルに期待する起動オフセット(単位：秒)
型	浮動小数点型
値の範囲	0..無限大
多重度	1
制限事項	－

RteExpectedTickDuration

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteOsInteraction/RteUsedOsActivation/RteExpectedTickDuration【rte_sws_9049_Conf】
概要	OS タスク，OS アラーム，もしくは OS スケジュールテーブルに期待する起動周期(単位：秒)
型	浮動小数点型
値の範囲	0..無限大
多重度	1
制限事項	－

3.14.11RteSwComponentInstance

コンテナ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance 【rte_sws_9005_Conf】
概要	SW-C プロトタイプのコンフィギュレーション.
多重度	0..*
パラメータ	RteSoftwareComponentInstanceRef
サブコンテナ	RteEventToTaskMapping
	RteExclusiveAreaImplementation
	RteExternalTriggerConfig(サポート対象外)
	RteInternalTriggerConfig(サポート対象外)
	RteNvRamAllocation(サポート対象外)
制限事項	－

RteSoftwareComponentInstanceRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance/ RteSoftwareComponentInstanceRef 【rte_sws_9004_Conf】
概要	コンフィギュレーション対象の SW-C プロトタイプ
型	*SW-C プロトタイプ(SwComponentPrototype)*への参照
値の範囲	－
多重度	0..1
制限事項	－

3.14.12RteEventToTaskMapping

コンテナ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance/ RteEventToTaskMapping【rte_sws_9020_Conf】
概要	RTE イベント(および RTE イベントから起動するランナブル)の OS タスクへのマッピング
多重度	0..*
パラメータ	RteActivationOffset
	RteEventRef
	RteImmediateRestart(サポート対象外)
	RteMappedToTaskRef
	RteOsSchedulePoint(サポート対象外)
	RtePositionInTask
	RteUsedOsAlarmRef
	RteUsedOsEventRef
	RteUsedOsSchTblExpiryPointRef(サポート対象外)
	RteVirtuallyMappedToTaskRef(サポート対象外)
サブコンテナ	－
制限事項	－

RteActivationOffset

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance/ RteEventToTaskMapping/ RteActivationOffset【rte_sws_9018_Conf】
概要	RTE イベントの起動オフセット(単位：秒)
型	浮動小数点型
値の範囲	0 ..無限大
多重度	0..1
制限事項	－

RteEventRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance/ RteEventToTaskMapping/ RteEventRef【rte_sws_9019_Conf】
概要	コンフィギュレーション対象の RTE イベントへの参照
型	**RTE イベント(RteEvent)**への参照
値の範囲	－
多重度	1
制限事項	－

RteMappedToTaskRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance/ RteEventToTaskMapping/ RteMappedToTaskRef【rte_sws_9021_Conf】
概要	RTE イベントのマッピング先の OS タスク
型	/AUTOSAR/EcucDefs/Os/OsTask への参照
値の範囲	－
多重度	0..1
制限事項	－

RtePositionInTask

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance/ RteEventToTaskMapping/ RtePositionInTask【rte_sws_9023_Conf】
概要	ランナブルの実行順番
型	整数型
値の範囲	0..65535
多重度	0..1
制限事項	－

RteUsedOsAlarmRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance/ RteEventToTaskMapping/ RteUsedOsAlarmRef【rte_sws_9024_Conf】
概要	RTE イベントのマッピング先 OS タスクを起動する OS アラーム
型	/AUTOSAR/EcucDefs/Os/OsAlarm への参照
値の範囲	－
多重度	0..1
制限事項	－

RteUsedOsEventRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance/ RteEventToTaskMapping/ RteUsedOsEventRef【rte_sws_9025_Conf】
概要	RTE イベントのマッピング先 OS イベント
型	/AUTOSAR/EcucDefs/Os/OsEvent への参照
値の範囲	－
多重度	0..1
制限事項	－

3.14.13RteExclusiveAreaImplementation

コンテナ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance/ RteExclusiveAreaImplementation【rte_sws_9030_Conf】
概要	排他エリアの実現メカニズム
多重度	0..*
パラメータ	RteExclusiveAreaImplMechanism
	RteExclusiveAreaOsResourceRef
	RteExclusiveAreaRef
	RteExclusiveAreaOsSpinlockRef
サブコンテナ	－
制限事項	－

RteExclusiveAreaImplMechanism

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance /RteExclusiveAreaImplementation/ RteExclusiveAreaImplMechanism【rte_sws_9029_Conf】
概要	排他エリアの実現メカニズム種別
型	列挙型
値の範囲	ALL_INTERRUPT_BLOCKING：全割込みの禁止 COOPERATIVE_RUNNABLE_PLACEMENT：協調ランナブル配置 OS_INTERRUPT_BLOCKING：OS 割込みの禁止 OS_RESOURCE：OS リソースの獲得 OS_SPINLOCK：OS スピンロックの獲得 NONE：排他なし
多重度	1
制限事項	本 RTE では，COOPERATIVE_RUNNABLE_PLACEMENT をサポートしない【nrte_sws_0048】.

RteExclusiveAreaOsResourceRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance /RteExclusiveAreaImplementation/ RteExclusiveAreaOsResourceRef【rte_sws_9031_Conf】
概要	排他エリアの実現のために使用する OS リソース
型	/AUTOSAR/EcucDefs/Os/OsResource への参照
値の範囲	－
多重度	0..1
制限事項	－

RteExclusiveAreaRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance /RteExclusiveAreaImplementation/ RteExclusiveAreaRef【rte_sws_9032_Conf】
概要	コンフィギュレーション対象の排他エリア
型	*排他エリア(ExclusiveArea)*への参照
値の範囲	－
多重度	1
制限事項	－

RteExclusiveAreaOsSpinlockRef

パラメータ名	/AUTOSAR/EcucDefs/Rte/RteSwComponentInstance /RteExclusiveAreaImplementation/ RteExclusiveAreaOsSpinlockRef【nrte_sws_0065_Conf】
概要	排他エリアの実現のために使用する OS リソース
型	/AUTOSAR/EcucDefs/Os/OsResource への参照
値の範囲	－
多重度	0..1
制限事項	－

4. リファレンス

4.1　RTE API 一覧

Std_ReturnType Rte_Write_<p>_<o>(IN　<data>)

Std_ReturnType Rte_Send_<p>_<o> (IN <data>)

Std_ReturnType Rte_Invalidate_<p>_<o> (void)

Std_ReturnType Rte_Feedback_<p>_<o> (void)

Std_ReturnType Rte_Read_<p>_<o> (OUT <data>)

Std_ReturnType Rte_Receive_<p>_<o> (OUT <data>)

Std_ReturnType Rte_Call_<p>_<o> ([IN | IN/OUT | OUT] <data_1>, ...[IN | IN/OUT | OUT] <data_n>)

<return> Rte_IrvRead_<re>_<o> (void)，または void Rte_IrvRead_<re>_<o> (OUT <data>)

void Rte_IrvWrite_<re>_<o>(IN <data>)

void Rte_Enter_<name>(void)

void Rte_Exit_<name>(void)

Std_ReturnType　Rte_Start(void)

Std_ReturnType　Rte_Stop(void)

void Rte_PartitionTerminated_<PID>(void)

void Rte_PartitionRestarting_<PID>(void)

Std_ReturnType　Rte_RestartPartition_<PID>(void)

4.2　RTE コールバック一覧

Rte_COMCbk_<sn>

Rte_COMCbkInv_<sn>

Rte_COMCbkRxTOut_<sn>

Rte_COMCbk_<sg>

Rte_COMCbkRxTOut_<sg>

Rte_COMCbkTAck_<sn>

Rte_COMCbkTErr_<sn>

Rte_COMCbkTxTOut_<sn>

Rte_COMCbkTAck_<sg>

Rte_COMCbkTErr_<sg>

Rte_COMCbkTxTOut_<sg>

4.3　SCHM API 一覧

void SchM_Enter_<bsnp>_<name>(void)

void SchM_Exit_<bsnp>_<name>(void)

Std_ReturnType SchM_Switch_<bsnp>_<name>(IN <mode>)

<return> SchM_Mode_<bsnp>_<name>(void)

void SchM_Init(void)

void SchM_Deinit(void)

4.4　データ型一覧

表 4-1 データ型一覧

データ型名	概要
Std_ReturnType	API 関数が返す「状態」と「エラー値」
<name>	RTE API で使用するデータ型 <name>は，以下のいずれかの実装データ型のショートネーム ・プリミティブ実装データ型 ・配列実装データ型 ・構造体実装データ型 ・共用体実装データ型 ・再定義実装データ型 ・ポインタ実装データ型
Rte_ModeType_ <ModeDeclarationGroup>	モード連携で使用するデータ型 <ModeDeclarationGroup>は，モード宣言グループのショートネーム

4.5　定数とマクロ一覧

4.5.1　定数一覧

表 4-2 定数一覧

定数名	概要
Rte_InitValue_<Port>_<DEPType>	データの初期値 <Port>は S/R 連携の送信側，もしくは受信側ポートのショートネーム．<DEPType>は S/R 連携の送信側，もしくは受信側データ要素のショートネーム．
RTE_E_<interface>_<error>	アプリケーションエラーの定数値 <interface>は C/S インタフェースのショートネーム， <error>はアプリケーションエラーのショートネーム．
<prefix><EnumLiteral>	データ型の列挙値 <EnumLiteral>は，*計算スケール(CompuScale)の計算定数(CompuConst)の文字列値(vt)*． <prefix>は，*計算方式(CompuMethod)*を使用する *AUTOSAR データ型(AutosarDataType)*を参照するインクルードデータ型セット*(IncludedDataTypeSet)*のリテラル接頭辞*(literalPrefix)*．
<prefix><DataType>_LowerLimit	データ型の下限値 <DataType>は，*アプリケーションプリミティブデータ型(ApplicationPrimitiveDataType)*のショートネーム． <prefix>は，*データ制約(dataConstr)*が属する *AUTOSAR データ型(AutosarDataType)*を参照するインクルードデータ型セット*(IncludedDataTypeSet)*のリテラル接頭辞*(literalPrefix)*．
<prefix><DataType>_UpperLimit	データ型の上限値 <DataType>，<prefix>はデータ型の下限値と同じ．
RTE_TRANSITION_ <ModeDeclarationGroup>	モード遷移中のステータス <ModeDeclarationGroup>は，モード宣言グループのショートネーム．
RTE_MODE_ <ModeDeclarationGroup>_ <ModeDeclaration>	モード宣言 <ModeDeclarationGroup>は，モード宣言グループのショートネーム． <ModeDeclaration>は，モード宣言のショートネーム．

4.5.2　マクロ一覧

表 4-3 マクロ一覧

マクロ名	概要
Rte_IsInfrastructureError(status)	API 返り値に，即時通信基盤エラーフラグのビットがセットされたときに 0 以外の値を返す.
Rte_HasOverlayedError(status)	API 返り値に，オーバーレイエラーフラグのビットがセットされたときに 0 以外の値を返す.
Rte_ApplicationError(status)	API 返り値のアプリケーションエラーコード(下位 6 ビット)を返す.

4.5.3　RTE エラーコード一覧

表 4-4 RTE エラーコード一覧

エラーコード	値
RTE_E_OK	0
RTE_E_INVALID	1
RTE_E_COM_STOPPED	128
RTE_E_TIMEOUT	129
RTE_E_LIMIT	130
RTE_E_NO_DATA	131
RTE_E_TRANSMIT_ACK	132
RTE_E_NEVER_RECEIVED	133
RTE_E_UNCONNECTED	134
RTE_E_IN_EXCLUSIVE_AREA	135
RTE_E_SEG_FAULT	136
RTE_E_DEV_DEFECT	191
RTE_E_LOST_DATA	64
RTE_E_MAX_AGE_EXCEEDED	64

4.5.4　SCHM エラーコード一覧

表 4-5 SCHM エラーコード一覧

エラーコード	値
SCHM_E_OK	0
SCHM_E_LIMIT	130
SCHM_E_NODATA	131
SCHM_E_TRANSMIT_ACK	132
SCHM_E_IN_EXCLUSIVE_AREA	135
SCHM_E_TIMEOUT	129
SCHM_E_DEV_DEFECT	191

変更履歴

Version	Date	Detail	Editor
1.0.0	2014/03/19	初版リリース	ESM
1.0.1	2014/06/27	nrte_sws_0237 を追加した.	ESM
1.0.1	2014/07/16	nrte_sws_0238 を追加した.	ESM
1.1.0	2014/11/21	RTE 機能拡張(※)に対して，以下の節を更新/追加. - 2.1.1 RTE 機能一覧 - 2.4.4 連携の実現方式 - 2.6.3 エクスキュータブル周期起動の実現方式 - 2.7.5 S/R 連携の実現方式 - 2.7.6 S/R 連携の設定 - 2.9 ランナブル間連携 - 2.10.4 排他エリアの設定 - 2.12.4 RTE タイプヘッダ - 2.12.10 RTE ソース - 2.13 コンフィギュレーション違反チェック - 2.14.5 ビルドサポート - 3.1.3 RTE 名前空間 - 3.1.5 API マッピング - 3.3 API データ型 - 3.5.1 初期値定数 - 3.6.5 トリガイベント - 3.8 RTE API - 3.10.1 COM コールバック - 3.13 依存インタフェース - 3.14 コンテナ - 4.1 RTE API 一覧 - 4.2 RTE コールバック一覧 - 4.4 データ型一覧 (※)RTE 機能拡張対象 - 複合データ型の追加 - AUTOSAR サービスの追加 - ジェネレートコードのインライン化 - IRV の追加	ESM
1.2.0	2015/2/28	A-COM のマルチコア対応に対して，以下の節を更新. - 2.2.11 コア	ESM

Version	Date	Detail	Editor
		nrte_sws_ext_0015 を変更 nrte_sws_ext_0033 を追加 - 2.4.4.3 ECU 間連携 nrte_sws_0334 を追加 irte_sws_0023 を追加 irte_sws_0024 を追加 COM 通信における性能改善対応に対して，以下の節を更新. - 2.4.4.3 ECU 間連携 nrte_sws_0010 を nrte_sws_0321 に変更	
1.2.0	2015/2/28	モード連携(SCHM)の機能追加に対して，以下の節を更新/追加. - 2.1.2 SCHM 機能一覧 - 2.5 エクスキュータブル動作管理 - 2.10 モード連携 - 2.14 コンフィギュレーション違反チェック - 3.3.8 モードのデータ型 - 3.11.3 SchM_Switch - 3.11.4 SchM_Mode - 3.14 コンテナ - 4. リファレンス	ESM
1.2.0	2015/2/28	データ型のアライメントに関する以下の仕様を削除. - 2.15.5 ビルドサポート rte_sws_7051	ESM
1.2.2	2015/6/12	参考文書を ATK2 外部仕様書に変更 - 1.2.2 参考文書 - 2.13.10.2 C/C++ 実装データ型に関する以下のモデル違反仕様を追加 - 2.14 コンフィギュレーション違反チェック nrte_sws_0346, nrte_sws_0349, nrte_sws_0350, nrte_sws_0351 モード連携に関する以下の API 仕様を変更 - 3.11.3 SchM_Switch - 3.11.4 SchM_Mode 対応している AUTOSAR 仕様の整合性調整のため，下記を修正	ESM

Version	Date	Detail	Editor
		- 2.4.2.1 連携のパターン - 2.5.2.1 ランナブル動作の種別 - 2.5.4.2 ランナブル起動 - 2.7.3.1 受信データセットの状態 - 2.7.6.3 ポート間の接続関係 - 2.7.6.4 データ要素間の接続関係 - 2.8.3.1 オペレーション呼出し - 2.8.5.7 ポート間の接続関係 - 2.10.2.1 モード連携の多重度 - 2.10.4.1 モード切替通知 - 2.15.1 RTE/SCHM コード生成フロー - 3.6.5.2 オペレーション呼出しイベント - 3.8.4 Rte_Read - 3.9.5 Rte_RestartPartition - 3.13 依存インタフェース - 3.14.7 RteGeneration 概要の誤字修正 - 3.14.10 RteUsedOsActivation	
1.3.0	2016/1/15	参考文書のバージョンを更新 - 1.2.2 参考文書 バックグラウンドイベント対応のため，下記の節を更新/追加. - 2.1 機能一覧 - 2.5 エクスキュータブル動作管理 - 2.7 エクスキュータブルバックグラウンド起動 - 2.13 ライフサイクル管理 - 2.15 コンフィギュレーション違反チェック - 3.6.5.2 バックグラウンドイベント 送信 ACK，データ送信完了イベント，データ受信イベント，データ受信エラーイベント対応のため，下記の節を更新/追加. - 2.5 エクスキュータブル動作管理 - 2.8 S/R 連携	ESM

Version	Date	Detail	Editor
		- 2.13.4.3 パーティションの停止，および再起動の通知 - 2.15 コンフィギュレーション違反チェック - 3.6.5.3 データ送信完了イベント - 3.6.5.4 データ受信イベント - 3.6.5.5 データ受信エラーイベント - 3.8.4 Rte_Feedback - 3.10.1.1 期待する COM コンフィギュレーション - 4.2 RTE コールバック一覧 BSW スケジューラブルのシグネチャの定義を修正 - 3.7.1 BSW スケジューラブルのシグネチャ 重複した仕様の削除 - nrte_sws_ext_0021 適用箇所のない仕様の削除 - nrte_sws_0249, nrte_sws_0250, nrte_sws_0251, nrte_sws_0252 その他、誤字の修正等	
1.4.0	2016/12/27	信頼関数におけるメモリチェック仕様について，以下の節を更新 - 2.4.4.2 パーティション間連携 nrte_sws_0378 を追加 - 2.15 コンフィグレーション違反チェック nrte_sws_0379, nrte_sws_0380, nrte_sws_0381, nrte_sws_0382 を追加 排他方式に関する仕様について，時間パーティショニング機能対応 OS の制約を反映し，以下の節を更新 - 2.8.5.8 排他方式 nrte_sws_ext_0035 を追加 排他方式を全割込みから OS 割込みへ変更 - 2.10.4.2 排他方式 nrte_sws_ext_0036 を追加 排他方式を全割込みから OS 割込みへ変更	ESM

Version	Date	Detail	Editor
		- 2.11.5.2 排他方式 排他方式を全割込みから OS 割込みへ変更 - 2.15 コンフィグレーション違反チェック nrte_sws_0383, nrte_sws_0384 を追加	