ASCET V6.2
AUTOSARユーザーズガイド

Document EC010201 V6.2 R01 JP – 10.2013

目次

図

コード

表

1 はじめに

本書は ASCET V6.2 がサポートする AUTOSAR 対応機能について説明するものです。

1.1 安全に関する注意事項

本製品を使用する際には、ユーザーの負傷やデバイスの損壊などを避けるため、製品の信頼性に関する免責条項(「ETAS Safety Advice - 安全上の注意事項」)、および下記の注意事項をよくお読みいただき、その指示に従ってください。

1.1.1 適切な製品の使用について

製品の不適切な使用や安全に関する注意事項に従わないことにより生じた一切の損害について、ETAS GmbH は責任を負いません。

1.1.2 安全に関する注意事項の記述書式

本書内に記述されている安全に関する注意事項には、下記の標準シンボルが併記されます。

安全に関する注意事項は以下の書式で記述されます。これらの情報は必ずよくお読みください。

警告!

中程度の危険性に関する注意事項です。記載事項を守らないと、重傷や生命の危険を招く可能性があります。

注意!

軽度の危険性に関する注意事項です。記載事項を守らないと、軽～中程度の負傷を招く危険性があります。

注記

物的損傷を招く可能性のある挙動についての説明です。

1.1.3 本製品に関する特殊な注意事項

本製品を安全に使用するには、一般的な注意事項に加え、以下の特殊な要件も守ってください。

- 本製品の準備や操作を行う前に、本製品を使用する環境が所定の条件を満たしていることを確認してください。各条件については、使用する PC やハードウェアのドキュメントを参照してください。

警告!

不適切に初期化された NVRAM 変数は、車両やテストベンチの予期しない挙動を生じさせる危険性があり、安全が脅かされる状況を招く恐れがあります。

ASCET-RP ターゲットの NVRAM 機能を使用する ASCET プロジェクトでは、ユーザー定義された初期化プロセス内で、すべての NV 変数の値がカレントプロジェクトに対して有効な状態になっているかを、個々の NV 変数単位および他の NV 変数との関連性においてチェックする必要があります。

データ保存に関する NVRAM の特性から、不適切な初期値が使用されることにより人体や装置が傷付けられる可能性のある環境内（車上やテストベンチなど）においてプロジェクトが使用される場合は、この要件を厳守してください。

さらに、製品 DVD に収められている ASCET V6.2 安全マニュアル（ASCET Safety Manual.pdf）に記載されている注意事項もよくお読みください。このドキュメントは製品インストール時に ETASManuals\ASCET V6.2 フォルダにコピーされ、また ETAS ホームページのダウンロードセンターからダウンロードすることもできます。

1.2 システム情報

ASCET 製品ファミリに含まれる各製品には、それぞれシミュレーションプロセッサとのインターフェース、サードパーティのソフトウェアパッケージとのインターフェース、さらに ASCET のリモートアクセスを行うためのインターフェース、といった機能が盛り込まれています。詳しくは『ASCET 入門ガイド』を参照してください。

現行バージョンの ASCET がサポートする AUTOSAR 対応機能を利用するには、以下の製品が必要です。

1. ASCET-MD
2. ASCET-SE
3. RTA-RTE（ASCET 製品ファミリには含まれません。詳しい情報は http://www.etas.com/ja/products/rta_rte.php をご覧ください）

1.3 ユーザーマニュアルについて

1.3.1 ユーザープロファイル

本書（ユーザーズガイド）は、ECU（自動車制御ユニット）の開発や適合作業の経験がある方を対象としています。本書をお読みいただくには、信号測定や ECU に関する技術についての専門的な知識が必要です。また ASCET についての知識と AUTOSAR についての知識（少なくとも基礎知識）も必要です。

ASCET についての基礎情報は『ASCET 入門ガイド』にまとめられています。

AUTOSAR についての知識が不十分である場合は、ASCET の AUTOSAR 対応機能を使用する前に AUTOSAR の概念について適切な情報を取得されることをお奨めします。

1.3.2 本書の構成

本書『ASCET AUTOSAR ユーザーズガイド』は以下の章で構成されています。

- 「**はじめに**」（本章）
  一般情報や、ユーザー、システムに関する情報です。

- 「**AUTOSAR の概要**」

  AUTOSAR について簡単に紹介しています。
- 「**ASCET によるソフトウェアコンポーネント開発**」

  This chapter describes how to configure ASCET for developing AUTOSAR software
  ASCET で AUTOSAR ソフトウェアコンポーネントを開発する際のコンフィギュレーション設定、ソフトウェアコンポーネントを作成するアプローチ、RTE ジェネレータの使用についての説明です。
- 「**データ型**」

  AUTOSAR で使用されるデータ型の紹介と、それを ASCET 内で使用する方法についての説明です。
- 「**インターフェース**」

  ASCET がサポートする AUTOSAR インターフェースについて紹介します。
- 「**ソフトウェアコンポーネント型**」

  ソフトウェアコンポーネント型とポートについて紹介し、それらを ASCET で使用する方法について説明します。
- 「**内部ビヘイビア（Internal Behavior）**」

  イベント用基本フレームワークとランナブルエンティティについての概要を説明し、さらに、RTE を設定してさまざまなタイプのランナブルエンティティとインターフェースとのインタラクションを実現する方法を紹介します。
- 「**モード**」

  ソフトウェアコンポーネントがランナブルエンティティの実行制御のために使用するアプリケーションモードを定義する方法を説明します。
- 「**ソフトウェアコンポーネントの実装**」

  ASCET でソフトウェアコンポーネントをモデリングして RTE が必要とするオブジェクトが宣言されるようにする方法と、RTE ジェネレータ生成する RTE API の使用方法を説明します。
- 「**お問い合わせ先**」

1.3.3 本書の使用法

本書は PDF 形式の電子ファイルで、いつでも PC の画面上で閲覧することができます。索引やテキスト検索、ハイパーリンクといった参照機能を用いて必要な情報に素早くアクセスできます。

ASCET をインストールすると、同時に AUTOSAR_UG_Tutorial というサンプルデータベースがインストールされます。本書で行う実習の内容は、このデータベースの Solutions というフォルダ内にモデリングされています。また ハードディスクの generated code_Solutions というサブディレクトリに ASCET が生成したコードが収められています。

表記について

ユーザーが実行するすべてのアクションは、いわゆる“Use-Case”形式で記述されています。つまり以下に示すように、操作を行う目標がタイトルとして最初に簡潔に定義され（例：「新しいコンポーネントを作成する」、「エレメントの名前を変更する」）、その下に、その目標を実現するために必要な操作手順が列挙され、必要に応じて ASCET のウィンドウやダイアログボックスのスクリーンショットが添付されています。

作業の目的（ゴール）：

- 1 番目の操作

  操作についての説明
- 2 番目の操作
- 3 番目の操作

表記上の規則

本書は以下の規則に従って表記されています。

`OCI_CANTxMessage msg0 =`	コードは灰色の背景の `Courier`（クーリエ）フォントで表記します。 コードの説明は、コメントとして表記します。コメントは一般的なコメントの表記法を使用しています。
File → Open を選択して….	メニューコマンドは **青い太字** で表記します。
OK をクリックします。	ボタンのラベルは**青い太字** で表記します。
<Enter>を押して…	キーボードのキーはブラケット◇で囲み、スモールキャップ（SMALL CAPS）形式で表記します。
“Open File” ダイアログボックスが開きます。	プログラムウィンドウ、ダイアログボックス、入力フィールド等のタイトルは、“ ”で囲んで表記します。
setup.exe ファイルを選択します。	リストボックス、プログラムコード、ファイル名、パス名等のテキスト文字列は、`Courier`（クーリエ）フォントで表記します。
ディストリビューション はサンプルポイントの 1 次元テーブルです	注意すべき個所や新出の用語等は**太字** 、または「 」で囲んで表記します。
OSEK グループ（http://www.osekvdx.org を参照）は各種標準規格を策定しています。	インターネットへのリンクは下線付きの青い文字で表記します。

特に重要な注意事項は、以下のように表記されています。

注記

ユーザー向けの重要な注意事項…

1.3.4 その他のドキュメント

ASCET がサポートする AUTOSAR 対応機能に関するより詳しい情報は、ASCET オンラインヘルプの“Software Component Editor”（「ソフトウェアコンポーネントエディタ」）および“AUTOSAR Interfaces”（「AUTOSAR インターフェース」）のセクションに記載されています。

また以下のユーザーマニュアルも各ソフトウェアとともにインストールされます。

- ASCET Getting Started（『ASCET 入門ガイド』）－ `ASCET V6.2 Getting Started.pdf`
- ASCET-SE User's Guide（『ASCET-SE ユーザーズガイド』）－ `ASCET-SE V6.2 Users Guide.pdf`
- RTA-RTE User's Guide、その他の RTA-RTE マニュアル：Windows **スタート** メニューの **ETAS** プログラムグループから **RTA-RTE** ***<x.y>*** **→ Documents →** ***<document>*** を選択してください。

これらのドキュメントは ETAS ウェブサイトのダウンロードセンターからダウンロードしていただくこともできます。

主要なドキュメントについては日本語版も用意されています。詳細はサポート窓口までお問い合わせください。

1.4 用語集

ASCET

ECU ソフトウェア開発ツール

ASCET-MD

ASCET Modeling and Design（ASCET モデリング／デザインツール）

AUTOSAR

Automotive **O**pen **S**ystem **Ar**chitecture（http://www.autosar.org/ 参照）

ARXML

AUTOSAR コンフィギュレーションの記述に使用される XML（E**X**tensive **M**arkup **L**anguage）

BSW

Basic **s**oft**w**are（基本ソフトウェア）― 通信、I/O など、一般的なすべてのソフトウェアコンポーネントが必要とする機能を提供するもの

CPU

Central **p**rocessing **u**nit（中央処理ユニット）

ECU

Embedded **C**ontrol **U**nit（組込み制御ユニット）

ERCOS^{EK}

OSEK 準拠の ETAS リアルタイムオペレーティングシステム

OS

Operating **s**ystem（オペレーティングシステム）

OSEK

独語：Arbeitskreis **O**ffene **S**ysteme fur die **E**lektronik im **K**raftfahrzeug（Open Systems and the Corresponding Interfaces for Automotive Electronics：自動車エレクトロニクス用オープンシステムおよびインターフェース）

RE

Runnable **e**ntity（ランナブルエンティティ）― 実行時に RTE によってトリガされる、SWC 内の一連のコードで、ASCET の「プロセス」にほぼ相当するもの

RTA-OSEK

ETAS リアルタイムオペレーティングシステム ― AUTOSAR-OS V1.0 (SC-1) と OSEK/VDX OS V2.2.3 の機能を提供し、MISRA に完全準拠

RTA-OS

ETAS リアルタイムオペレーティングシステム ― AUTOSAR R3.0 OS と OSEK/VDX OS V2.2.3 の機能を提供し、MISRA に完全準拠

RTA-RTE

ETAS の AUTOSAR **r**un**t**ime **e**nvironment（AUTOSAR 実行環境）

RTE

AUTOSAR **r**untime **e**nvironment（AUTOSAR 実行環境）― ソフトウェアコンポーネント、基本ソフトウェア、オペレーティングシステム間のインターフェースを提供するもの

SWC

AUTOSAR **s**oft**w**are **c**omponent（AUTOSAR ソフトウェアコンポーネント）― AUTOSAR における分割不可能な最小単位のソフトウェアコンポーネント

UUID

Universally **U**nique **Id**entifier（統一モデリング言語）

VFB

Virtual **F**unction **B**us（仮想ファンクションバス）

2 AUTOSAR の概要

昨今の車両プロジェクトは、複数の電子制御ユニット（ECU）やネットワーク、そしてさまざまなソフトウェアアーキテクチャから構成され、そこに複数のサプライヤから提供されたソフトウェアコンポーネントを統合するには非常に大きな労力を要します。このような環境においては、プロジェクト間での自動車用組み込みソフトウェアの再利用性が明らかに制限されるだけでなく、十分な機能性が実装された実証済みソフトウェアを提供するには、さらに余分な工程が必要になります。

AUTOSAR パートナーシップは、特に基本的なシステム機能と機能的インターフェースを標準化することにより、自動車エレクトロニクス向けソフトウェアの共同開発の簡略化、開発コストの低減、製品化までの時間の短縮、および品質の向上を目標とし、安全関連システムの設計に必要なメカニズムを提供しています。

これらの目標に到達するために、AUTOSAR では自動車用組み込みソフトウェアのアーキテクチャを定義しています。このアーキテクチャでは、各アプリケーションの機能（「ファンクション」）を実装する「ソフトウェアコンポーネント」が柔軟かつ効率的に扱われ、ECU に依存しないソフトウェアコンポーネントの再利用、交換、規模変更、統合を容易に行うことができます。

以下の項では、AUTOSAR 対応のアプリケーションソフトウェアコンポーネントの開発方法について簡単に説明します。詳細については、AUTOSAR ウェブサイト（http://www.autosar.org/）の資料を参照してください。

2.1 AUTOSARの基本的アプローチ

AUTOSAR において「アプリケーションソフトウェア」とは、さまざまな車両ファンクションを指します。各アプリケーションは 1 つまたは複数の「ソフトウェアコンポーネント」（SWC）で構成されます。SWC は特定の CPU やロケーションに依存しないように作られています。システムコンフィギュレーションを設定する際、このソフトウェアコンポーネントを任意の ECU にマッピングすることができます。

SWC 環境を抽象化したものは仮想ファンクションバス（VFB：**V**irtual **F**unction **B**us）と呼ばれます。実際の AUTOSAR 対応の ECU では、VFB のマッピングは ECU に依存する特定のインプリメンテーションのプラットフォームソフトウェアにより実現されます。AUTOSAR プラットフォームソフトウェアはランタイム環境（RTE）および基本ソフトウェア（BSW）という 2 つの機能領域に大きく分けられます。

BSW は通信や I/O の機能だけでなく、診断やエラーレポーティング、不揮発メモリ管理など、ソフトウェアコンポーネントが使用する機能を提供します。

アプリケーション SWC から BSW への直接アクセスは行えません。つまり、コンポーネントからオペレーティングシステムや通信サービスなどに直接アクセスすることはできません。代わりに、「ランタイム環境」（RTE：**R**un**t**ime **E**nvironment）がソフトウェアコンポーネント、BSW モジュール、およびオペレーティングシステム（OS）の間のインターフェースとなります。SWC の相互接続においては、RTE は電話の配電盤のように機能します。これは、接続するコンポーネント同士が単一 ECU に常駐する場合も、車両バスで接続された複数のネットワーク ECU に常駐する場合も同じです。

AUTOSAR では、OS は RTE を通じて SWC のランナブルエンティティを呼び出します。また基本ソフトウェアにとって RTE と OS は、アプリケーションソフトウェアの実行制御における重要なモジュールです。

ETAS は 10 余年にわたり自動車業界に自動車オペレーティングシステム（ERCOS^{EK} および RTA-OSEK）を提供しています。主要な AUTOSAR ソフトウェアモジュールをサポートする RTA-RTE AUTOSAR Runtime Environment および RTA-OS AUTOSAR Operating System を加え、RTA 製品ラインはますます充実したものになりました。これらの製品の AUTOSAR 対応機能により、サードパーティ製の基本ソフトウェアモジュールも RTA-RTE や RTA-OSEK とシームレスに統合できます。

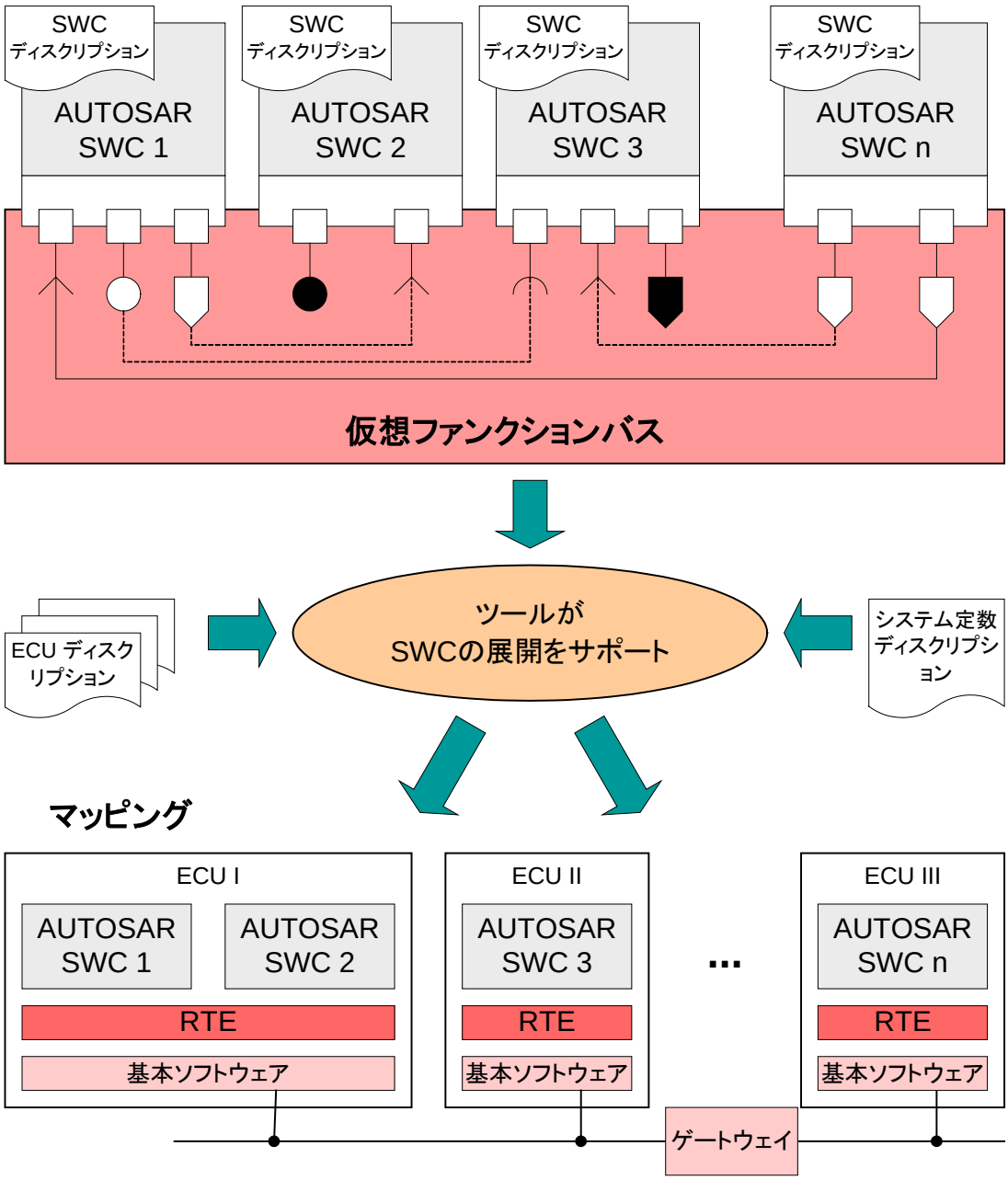


図 **1**: ランタイム環境（RTE）および基本ソフトウェア（Basic Software）として実装された仮想ファンクションバス（VFB）により実現される AUTOSAR ソフトウェアコンポーネント（SWC）間の通信

2.2 AUTOSARオーサリングツールとは

「AUTOSAR オーサリングツール」とは、以下のような AUTOSAR の各種ディスクリプションの翻訳、加工、生成をサポートするソフトウェアツールです。

- ソフトウェアコンポーネントディスクリプション －以下のものに関する記述です。
  - ソフトウェアコンポーネントが提供／要求する、処理とデータエレメント
  - ソフトウェアコンポーネントからインフラストラクチャへの要求事項
  - ソフトウェアコンポーネントに必要なリソース（メモリ、CPU 時間など）
  - ソフトウェアコンポーネントの具体的な実装情報
- システム制約ディスクリプション － すべてのシステム情報、および ECU 間で一致していなければならない情報に関する記述です。

- ECU ディスクリプション ― 各 ECU のリソースとコンフィギュレーションに関する記述です。

AUTOSAR SWC は車両ネットワーク内の CPU にもロケーションにも依存しない、汎用アプリケーションレベルのコンポーネントです。システムコンフィギュレーションの設定により、システム設計者が定義した制約条件に基づいて各 SWC を任意の ECU にマッピングすることができます。この AUTOSAR ソフトウェアコンポーネントは AUTOSAR システムにおいて分割不可能な「アトミックユニット(=最小単位)」であるため、1 つのコンポーネントを複数の ECU 上にまたがってマッピングすることはできません。

SWC を作成するには、まずコンポーネント型(「SWC 型」)を定義する必要があります。SWC 型は SWC の固定的な特性(ポート名、ポートのインターフェース種別、SWC のビヘイビアなど)を明らかにするもので、各 SWC 型にはシステム内で一意の名前を付ける必要があります。1 つの SWC は以下のもので構成されます。

- コンポーネントのインフラストラクチャの設定方法を示す完全な SWC ディスクリプション
- C コード形式の機能記述を含む SWC インプリメンテーション

SWC を使用可能な状態にするには、コンフィギュレーション設定時にその SWC をインスタンス化する必要があります。「型」と「インスタンス」の関係は、従来のプログラミング言語における「型」と「変数」の関係に相当します。つまり、アプリケーション規模で一意の型名(SWC 型)を定義し、その型の変数(1 つまたは複数の SWC インスタンス)を一意の名前で宣言します。

VFB モデルにおいて、ソフトウェアコンポーネント間のインタラクションは、インターフェース別の「ポート」を通じて行われます。「インターフェース」は、通信される情報や通信のセマンティクスを制御し、ポートは、SWC からインターフェースへのアクセスを提供します。ポートとインターフェースの組み合わせは「AUTOSAR インターフェース」と呼ばれます。

「ランナブルエンティティ」は、実行時に RTE によりトリガされる SWC 内のコード部分です(2.3 項「ランタイム環境(RTE)とは」を参照)。

1 つのソフトウェアコンポーネントは、実行時に RTE がアクセスできる 1 つまたは複数のランナブルエンティティで構成されます。ランナブルエンティティは以下のイベントによりトリガされます。

- タイミングイベント ― 周期的スケジューリングイベント(周期的なタイマのチックなど)です。ランナブルエンティティは周期的実行のためのエントリポイントを提供します。
- データ受信イベント ― R ポートにおけるデータ受信によりトリガされるイベントです。

AUTOSAR ランナブルエンティティはいくつかのカテゴリに分類されています。ASCET はカテゴリ 1 のランナブルエンティティをサポートしています。

ランナブルエンティティを実行するためには、そのランナブルエンティティを AUTOSAR オペレーティングシステムのタスクに割り当てておく必要があります。

各 AUTOSAR エレメントは標準化された XML ファイル(いわゆる ARXML 形式)内で相互参照します。ARXML のフォーマットは AUTOSAR のリリースバージョンにより若干異なりますが、AUTOSAR オーサリングツールにはこの ARXML ディスクリプションの翻訳、生成、変更を行う機能が不可欠です。

注記

本書で紹介している ARXML の例は、AUTOSAR リリースバージョン 3.1.2 を使用して生成したものです。

2.3 ランタイム環境(RTE)とは

VFB は「抽象化」によりコンポーネントの再利用を可能にするもので、実行時には、VFB の抽象化を機能させるためのメカニズムがランタイム環境(RTE)によって提供されます。単純に言い換えると「VFB を実装したものが RTE である」と表現することができますが、RTE は、ソフトウェアコンポーネントについて以下の情報を実現するために必要なインターフェースとインフラストラクチャを提供する必要があります。

1. 1 つの ECU への参照を行わずに実装できること

2. 「システム統合モデル」のコンポーネントとして認識された後は、アプリケーションソフトウェア自体を変更しなくても ECU および広範囲の車両ネットワークに統合できること

さらに具体的に言うと、RTE は以下のことを実現する必要があります。

- ソフトウェアコンポーネント用の通信インフラストラクチャの提供

  同一 ECU 上のソフトウェアコンポーネント間の通信（ECU 内通信）と、異なる ECU 上のソフトウェアコンポーネント間の通信（ECU 間通信）の両方が含まれます。
- ソフトウェアコンポーネントのリアルタイムスケジューリングの管理

  一般的には、SWC の各ランナブルエンティティを、設計時に定義された時間的制約条件に基づき、オペレーティングシステムが提供するタスクにマッピングします。

RTE により実装された「抽象化」の下では、アプリケーションソフトウェアコンポーネントは基本ソフトウェアに直接アクセスできません。つまり、アプリケーションソフトウェアコンポーネントはオペレーティングシステムや通信サービスなどを直接利用できないので、RTE はそのようなサービスの上に「抽象化されたもの」を用意する必要があります。この「抽象化されたもの」は、ソフトウェアコンポーネントのロケーションにかかわらず不変であることが必要不可欠です。ソフトウェアコンポーネント間のすべてのインタラクションは、標準化された RTE インターフェースの呼び出しにより行われます。

さらに RTE は、1 つまたは複数の ECU 上の SWC で構成される定義済みアーキテクチャを具体的に実現するためにも用いられます。効率化のため、所定のアーキテクチャを実現するために実装される RTE は、ECU ごとにビルド時に決定されます。標準化された RTE インターフェースが RTE 生成ツールにより自動的に実装され、コンポーネントのインタラクションや配置に応じたインターフェースが確実に生成されます。

たとえば、2 つのソフトウェアコンポーネントが同じ ECU 上に常駐している場合は、それらの間で内部 ECU 通信を使用できますが、一方のコンポーネントを別の ECU に移動した場合は、車両ネットワーク経由で通信する必要があります。

アプリケーションソフトウェアコンポーネントの側から見ると、生成された RTE は以下のようにして各種 ECU の基本ソフトウェアの相違をカプセル化します。

- 一貫性のあるインターフェースをソフトウェアコンポーネントに提供し、ソフトウェアコンポーネントの再利用性を確保します。一度設計され作成されたソフトウェアコンポーネントを何度も利用できるようになります。
- このインターフェースを、VFB の抽象化設計において実装された AUTOSAR 基本ソフトウェアにバインドします。

2.4 ビヘイビアモデリングツールとは

AUTOSAR ビヘイビアモデリングツールでは、「ビヘイビアモデリング言語」を使用して、AUTOSAR 準拠の車両ファンクションの機能的ビヘイビア（機能的挙動）を定義して実装することができます。

ビヘイビアモデリング言語は、主として機能的ビヘイビア仕様やファンクション／システム設計を掌握するために使用される表記法です。一般的にビヘイビアモデリング言語はグラフィック表記を採用し、「実行可能」なものと見なされます。つまり、ビヘイビアモデルをそのままシミュレーションエンジンで実行できるほど精密なセマンティックスを備えています。このようなセマンティックの精密さにより、ファンクションモデルを C などのプログラミング言語のソースコードに変換することも可能です。

ASCET をビヘイビアモデリングツールとして使用する場合、アプリケーションソフトウェアコンポーネントの内部ビヘイビアはブロックダイアグラムエディタで記述します。内部ビヘイビアを構成するのは、変数、メッセージ、パラメータ、クラスインスタンス、モジュールです。AUTOSAR ランナブルエンティティは、メソッド呼び出しとプロセスシーケンスによりシームレスに実装できます。

AUTOSAR のコンセプトの多くは ASCET のインターフェース定義によく似ているため、既存の ASCET モデルを AUTOSAR に容易に適応させることができます。ASCET は、既存ソフトウェアモジュールのインターフェースを再構築して AUTOSAR 対応のものにするための機能を備えているため、既存アプリケーションの作り直しに必要な作業工数を削減することができます。

3 ASCET によるソフトウェアコンポーネント開発

3.1 ASCETのコンフィギュレーション設定

本項では、ASCET のコンフィギュレーションを AUTOSAR ソフトウェアコンポーネント開発用に設定する方法について簡単に説明します。ASCET の詳しい操作方法については『ASCET 入門ガイド』（ASCET Getting Started）と ASCET オンラインヘルプを参照してください。

3.1.1 AUTOSARコンポーネントの作成に関する設定

ASCET のユーザーオプションには、AUTOSAR コンポーネント作成用の設定オプションが含まれています。

AUTOSAR コンポーネントの作成を有効にする:

- ASCET コンポーネントマネージャの **Tools → Options** を選択します。“Options”ダイアログボックスが開きます。
- ダイアログボックス左側のツリービューの“Modeling”ノードにある **Enable Creation of AUTOSAR components** チェックボックスをクリックしてオンにします。
- **OK** をクリックします。

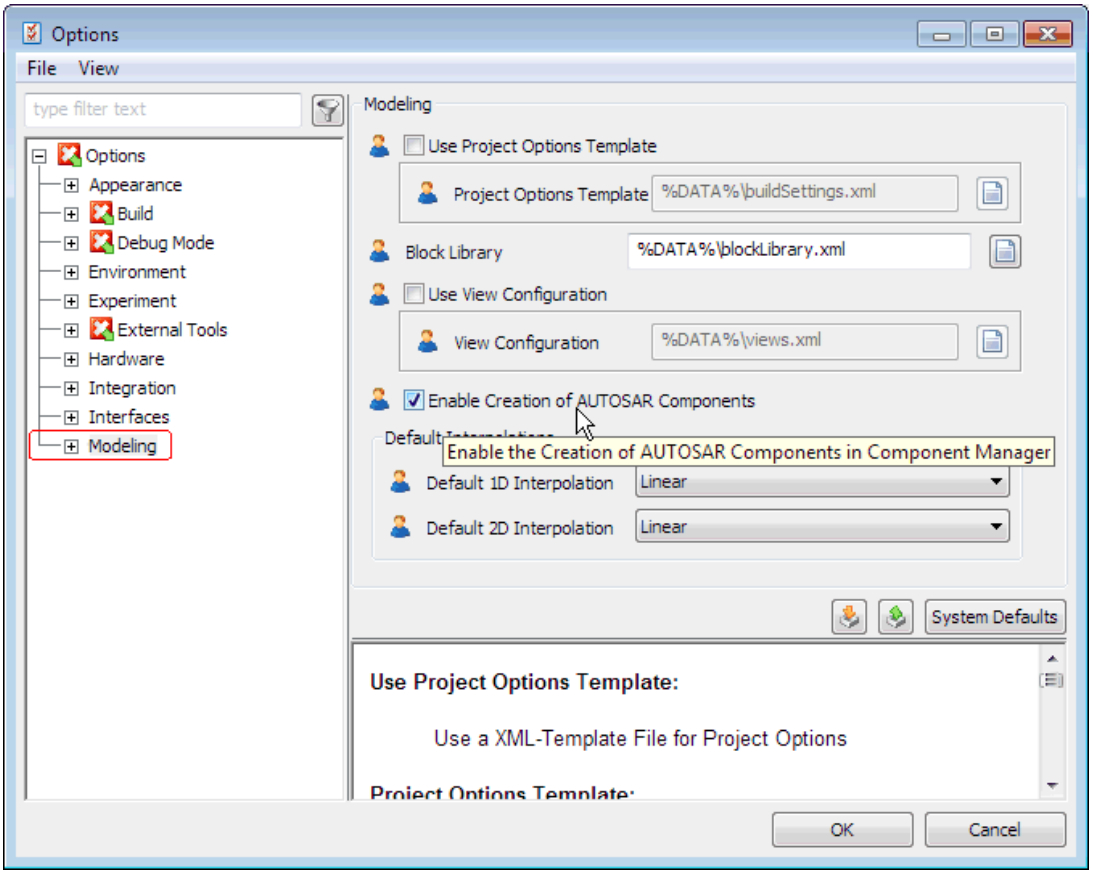


図 2: AUTOSAR コンポーネントの作成を有効にする

3.1.2 AUTOSAR用コード生成に関する設定

ASCET における「プロジェクト」は、1 つの完全なソフトウェアシステムを構成するメインユニットです。プロジェクトのコンテキストにおいて、変換式や実装型などが定義されます。

プロジェクトを作成する:

- コンポーネントマネージャで **Insert → Project** を選択するか、または **Insert Project** ボタンをクリックして、新規プロジェクトを作成します。
- このプロジェクトの名前を `ARProject` にします。
- **Edit → Open Component** を選択するか、または `ARProject` をダブルクリックします。
  プロジェクトエディタが開きます。

コード生成オプションを AUTOSAR 用に設定する:

- プロジェクトエディタで **File → Properties** を選択するか、または **Project Properties** ボタンをクリックします。
  “Project Properties”ダイアログボックスが開きます。
- “Build ノード”で以下のオプションを選択します。
  - **Target**: `ANSI-C`

    注記

    ASCET V6.2 では、AUTOSAR コード生成に使用できるターゲットは `ANSI-C` のみです。

  - **Operating System**: `RTE-AUTOSAR x.y.z`

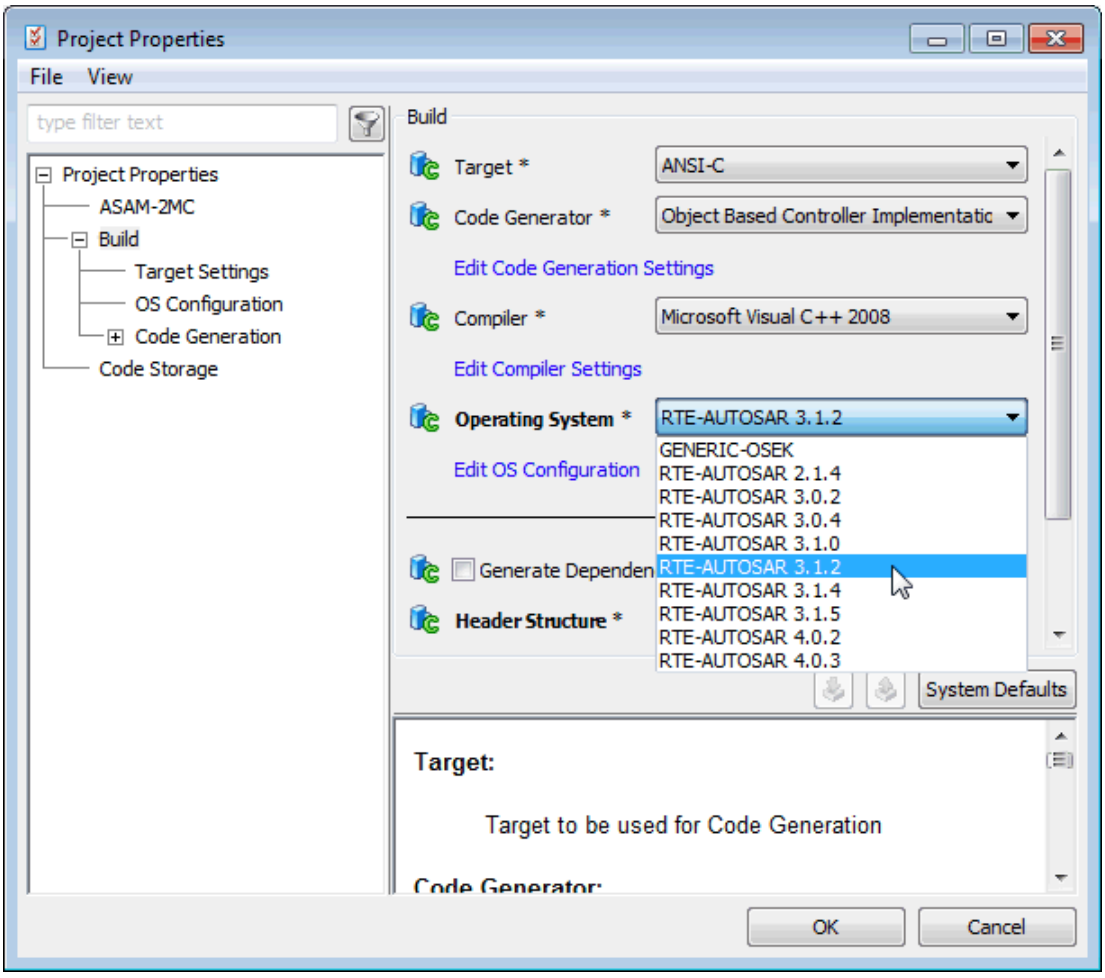


図 3: AUTOSAR プロジェクトのプロジェクト設定

注記

ASCET V6.2 は AUTOSAR の以下のリリースをサポートしています。

2.1.4, 3.0.2, 3.0.4, 3.1.0, 3.1.2, 3.1.4, 3.1.5, 4.0.2, 4.0.3.

- “Code Generation”ノードの **Casting** コンボボックスで `MISRA compliant` を選択します。
  他のキャストオプションは、AUTOSAR には適していません。

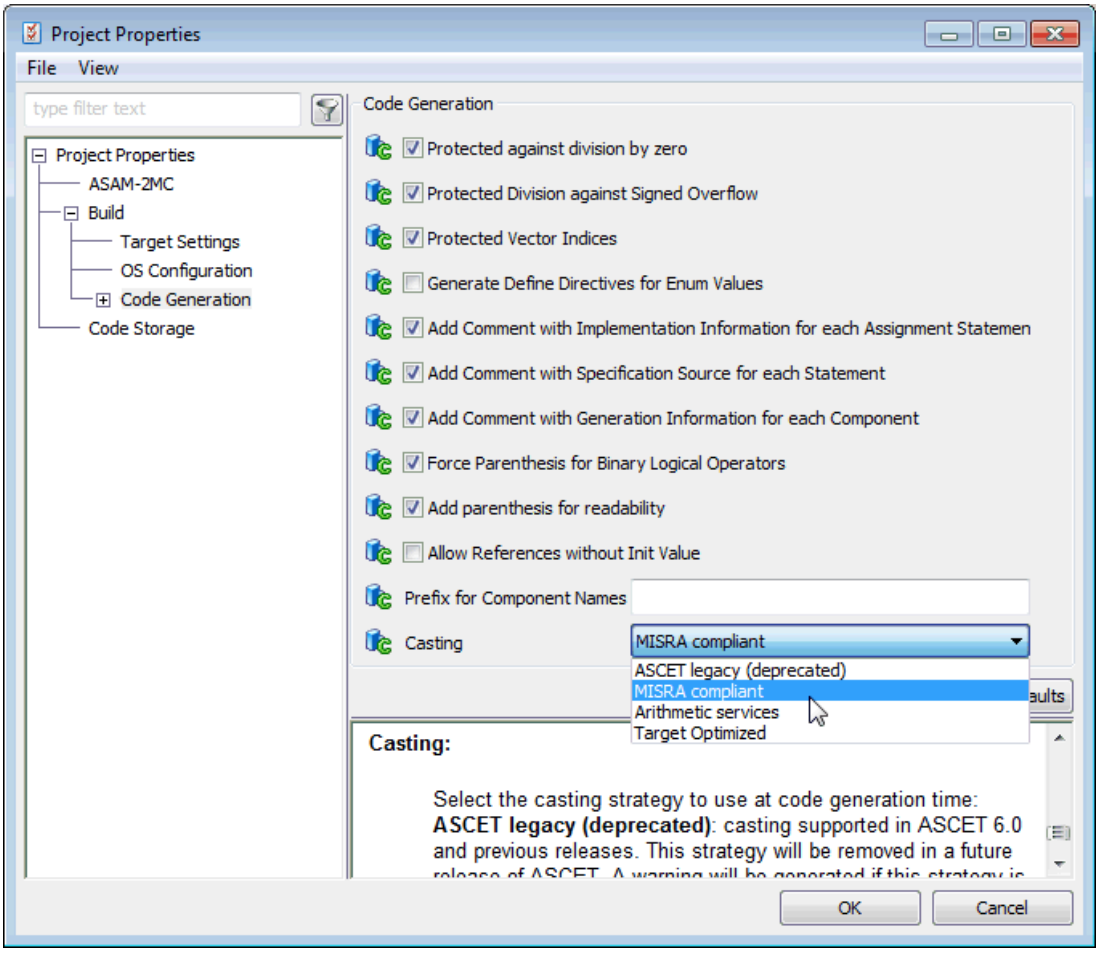


図 4: AUTOSAR プロジェクトのプロジェクト設定: MISRA 対応キャストモード

メモリセクション定義ファイルを定義する:

AUTOSAR プロジェクトのコード生成を行う際、ASCET は XML コンフィギュレーションファイルからメモリセクション情報を読み込みます。このファイルはプロジェクトプロパティの“OS Configuration”ノードで指定します（図 5 参照）。

- “Project Properties”ダイアログボックスの“OS Configuration”ノードを開きます。
- **Memory Sections Configuration File** フィールドで、メモリセクション定義を含む XML ファイルのパスと名前を指定します。

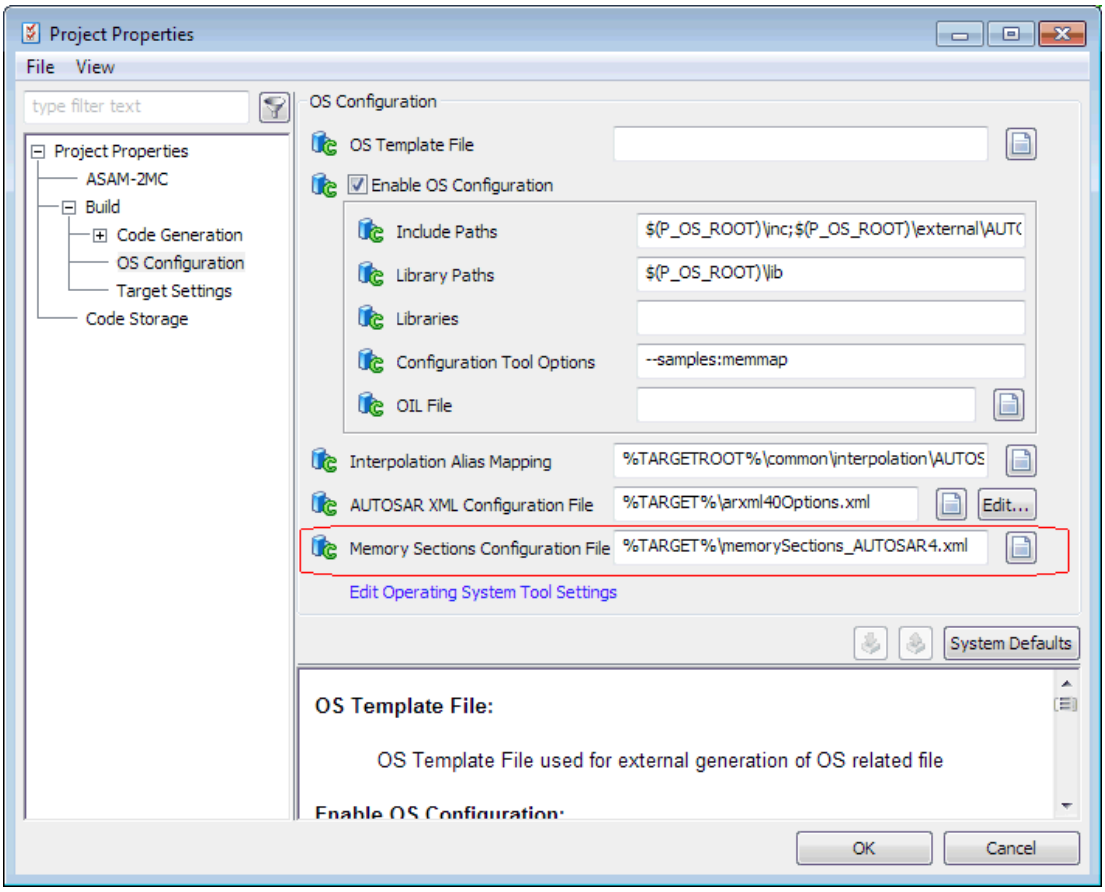


図 5: AUTOSAR R4.0.*プロジェクト用 OS コンフィギュレーション設定

ASCET には、`memorySections_Autosar.xml`(AUTOSAR R3.1.5 以前のバージョン用)および `memorySections_Autosar4.xml`(AUTOSAR R4.0.*用)という 2 つのサンプルファイルが付属しています。ここでは、選択されているオペレーティングシステム RTE-AUTOSAR *(3.1.2 項「AUTOSAR 用コード生成に関する設定」を参照)に適した `memorySelecions_*.xml` ファイルがあらかじめ選択されています。

3.1.3 AUTOSAR XML出力に関する設定

“Project Properties”ダイアログボックスでは、AUTOSAR XML コンフィギュレーションファイルに関する設定(パッケージ名やショートネームの設定)を行って、出力ファイルを指定することができます。

AUTOSAR XML(ARXML)出力を設定する:

- “Project Properties”ダイアログボックスの“OS Configuration”ノードを開きます。
- “AUTOSAR XML Configuration File”フィールドでコンフィギュレーションファイルの名前を選択(または入力)します。

  デフォルトでは、各 AUTOSAR バージョン(Rx.y)ごとにそれぞれ異なるコンフィギュレーションファイルが使用されます。AUTOSAR バージョンに応じて ARXML 設定が異なるため、このデフォルト設定を変更しないようにすることをお勧めします。
- **Edit** ボタンをクリックして“ARXML Configuration Settings”ダイアログボックスを開きます。

  このダイアログボックスには、AUTOSAR XML の生成に関する一連のオプションが含まれています。各オプションは、 表 1 に示されるように、複数のカテゴリに分類されます。

カテゴリ(ノード)	内容
Package Templates	ARXML パッケージ名を以下の構文で指定するためのテンプレートです。 `/<Root-Package>/<Sub-Package>/.../<Short-Name>` 各テンプレートごとに専用のオプションを使用します。
Short Name Templates	ARXML ショートネームを定義するためのテンプレートです。 各テンプレートごとに専用のオプションを使用します。
Filename Templates	関連するパッケージが生成されるファイル名を定義するためのテンプレートです。 各テンプレートごとに専用のオプションを使用します。
Miscellaneous	その他のオプションを定義します。テンプレートなど、ARXML の生成に関するオプションが含まれます。

表 1: 生成される ARXML コードに関する設定のカテゴリ(AUTOSAR バージョンにより異なる)

- 必要に応じて各カテゴリに含まれるオプションを設定します。
  "ARXML Configuration Settings"ダイアログボックスの下部に、各オプションについての詳しい説明が表示されます。
- **OK** をクリックして設定を確定し、"ARXML Configuration Settings"ダイアログボックスを閉じます。

注記

"ARXML Configuration Settings"ダイアログボックスの設定内容は、"Project Properties"ダイアログボックスで **Cancel** をクリックした場合も保持されます。

3.1.4 コード生成

AUTOSAR プロジェクトには、必ず 1 つの AUTOSAR ソフトウェアコンポーネントを定義し、前項で説明したプロジェクト設定を行う必要があります。このプロジェクトのコード生成を実行すると、ASCET は AUTOSAR XML ディスクリプションファイル(`*.arxml` ファイル)とそれに対応する C コードを作成します。生成された C コード内では、RTE を実装する AUTOSAR API マクロが使用されます。

AUTOSAR ソフトウェアコンポーネントを作成する:

- コンポーネントマネージャの **Insert → AUTOSAR → Software Component** を選択します。
- このソフトウェアコンポーネントの名前を Swc にします。

AUTOSAR ソフトウェアコンポーネントをプロジェクトに挿入する:

- プロジェクト `ARProject` を開きます。
  プロジェクトエディタが開きます。
- プロジェクトエディタで **Insert → Component** を選択します。
  "Select Item..."ダイアログボックスが開きます。
- "1 Database"または"1 Workspace"フィールドで、コンポーネント Swc を選択します。

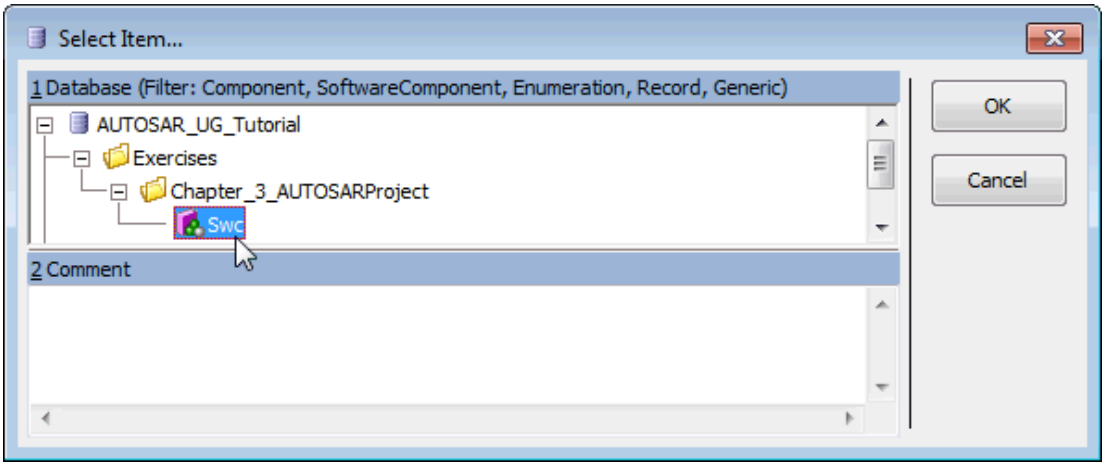


図 6: プロジェクト ARProject のアイテム Swc を選択する

- **OK** をクリックして“Select Item...”ダイアログボックスを閉じ、Swc をプロジェクトに挿入します。

  “Properties for Complex Element”ダイアログボックスが開きます。ここでは Swc のインスタンス名やコメントなどを入力できます。
- **OK** をクリックしてデフォルトの名前とコメントをそのまま使用します。

プロジェクトのコードを生成する:

- プロジェクトエディタで **Build → Touch →Recursive** を選択し、続いて **File → Export → Generated code → Recursive**.

> **注記**
>
> ASCET V6.2 以降では、**Build → Touch → Recursive** の後に **Build → Generate Code** を選択する必要はありません。

“Path to export Items”ダイアログボックスが開きます。デフォルトとして Cgen（ASCET のコード生成ディレクトリ）が選択されています。

> **注記**
>
> ASCET インストールディレクトリ下の CGen ディレクトリは、コードジェネレータの中間結果を保存するための一時ディレクトリです。コードをこのディレクトリに保存することは推奨しません。

- 生成済みコードのエクスポート先を指定します。

  ここでは ASCET のカレントデータベースのサブディレクトリ（例、`C:\ETASData\ASCET6.2\Database\AUTOSAR_UG_Tutorial`）などを指定します。

  `ARProject` プロジェクトには空の AUTOSAR ソフトウェアコンポーネント Swc が含まれているので、生成されるファイルは以下のようになります。

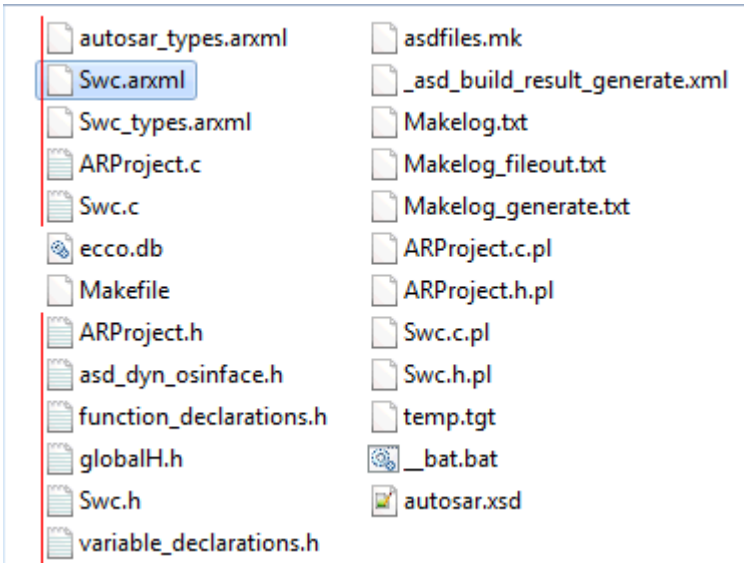


図 7: ASCET が生成したプロジェクト ARProject 用コード (*.arxml、*.c、*.h ファイル)

3.2 ソフトウェアコンポーネント作成のアプローチ

ASCET における AUTOSAR ソフトウェアコンポーネントの開発は、「トップダウン」と「ボトムアップ」という 2 通りのアプローチで行うことができます。

トップダウンアプローチでは、ソフトウェアアーキテクチャをオーサリングツールで記述し、ソフトウェアコンポーネントを実装するためのビヘイビアモデリングツールとして ASCET を使用します。

ボトムアップアプローチでは、ASCET をビヘイビアモデリングツールとしてだけでなく AUTOSAR ソフトウェアコンポーネントを記述するオーサリングツールとしても使用します。

3.2.1 トップダウンアプローチ

トップダウンアプローチでは、AUTOSAR ソフトウェアコンポーネントの作成は以下の 2 ステップで行います。

1. 第 1 ステップでコンポーネントのインターフェースを定義します。インターフェースはオーサリングツールで記述し、ARXML に変換します。ARXML ファイル(1 つまたは複数)はコンポーネント API ジェネレータに渡され、そこでインターフェースディスクリプションがヘッダファイルに変換されます。基本的に、コンポーネント API ジェネレータは RTE ジェネレータの「コントラクトフェーズ部分」に当たります(3.3.1 項「コントラクトフェーズ」を参照)。
2. 第 2 ステップでは、ARXML ファイル(1 つまたは複数)を ASCET にインポートし、アプリケーションソフトウェアコンポーネント開発者が、ヘッダファイルに定義されたインターフェースを使用する C ファイルの形式で内部ビヘイビアを記述します。このようにしてソフトウェアコンポーネントの*.h ファイルと*.c ファイルが定義され、コンパイル可能な状態になります。

トップダウンアプローチで重要な役割を果たしているのは ARXML インポータです。これについて次項以降で説明します。

ARXML インポータ

ソフトウェアコンポーネントの ARXML ディスクリプションは、「AUTOSAR to ASCET インポータ」により ASCET にインポートされます。インポート時には、まずこのツールが、ソフトウェアコンポーネントの記述に必要なすべての情報(AUTOSAR 型、インターフェース、ソフトウェアコンポーネント型)を含んだ ARXML ファイルを ASCET 独自の XML 形式(「AMD 形式」)に変換し、ASCET がその AMD ファイルを現在開いているデータベースまたはワークスペースにインポートします。

「AUTOSAR to ASCET インポータ」を起動するには、コンポーネントマネージャで **Tools → AUTOSAR to ASCET Converter** を選択します。詳しくは『AUTOSAR to ASCET Importer User's Guide』を参照してください。

ARXML ファイルのインポートには、標準のインポート用メニューコマンドを使用します。

ARXML を ASCET にインポートする:

- コンポーネントマネージャで **File → Import** を選択します。
  "Select Import File"ダイアログボックスが開きます。
- インポートする ARXML ファイル(1 つまたは複数)を選択して **OK** をクリックします。
  選択されたファイルが現在開いているデータベースまたはワークスペースにインポートされます。

ARXML インポータにおける UUID 属性の使用

UUID(Universally Unique Identifier)は ARXML 定義内のオプショナルフィールドで、ほとんどのオーサリングツールがサポートしています。ASCET も AMD 形式の UUID をサポートしているので、AUTOSAR ツールチェーンに ASCET を容易に統合することができます。ただし現時点では、UUID 属性とともにインポートされたエレメントについては ASCET が生成する ARXML に UUID 属性が含まれますが、そうでないエレメントは UUID 属性が空になります。

UUID は主に、ARXML ファイルをインポートする際に ASCET データベース／ワークスペースの既存コンポーネントを識別するために使用されます。UUID 属性の使用は明示的に有効にする必要があります。

コンポーネント識別用に UUID を使用する:

- コンポーネントマネージャで **Tools → Options** を選択します。
  ASCET オプションダイアログボックスが開きます。
- "Interfaces/Import"ノードを開きます。
- **Use UUIDs for Identification** オプションをオンにします。
- **OK** をクリックして設定を確定し、ASCET オプションダイアログボックスを閉じます。

  Use UUIDs for Identification オプションは、"Select Import File"ダイアログボックスでも設定できます(図 8)。

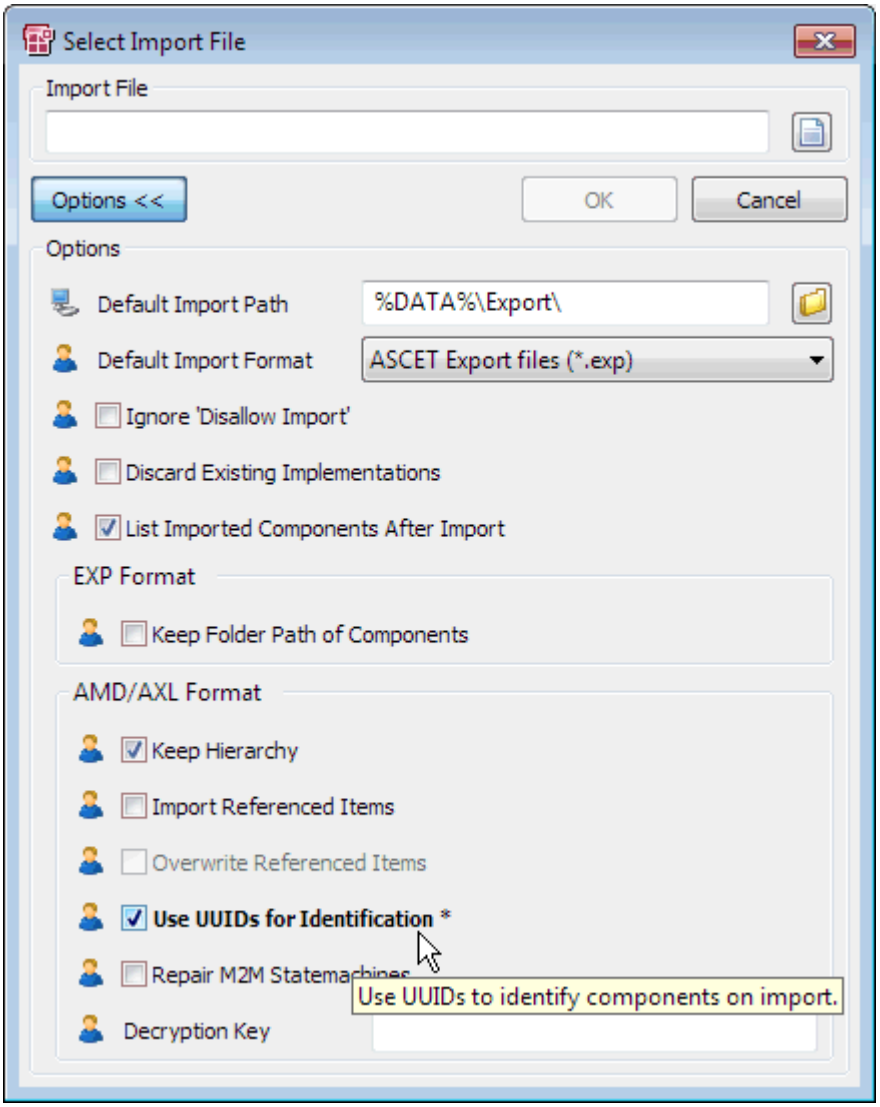


図 8: インポート時にコンポーネント識別用の UUID を使用する

3.2.2 ボトムアップアプローチ

ASCETは、アプリケーションソフトウェアコンポーネントのオーサリングツールとしてもビヘイビアモデリングツールとしても使用できます。ボトムアップアプローチでは、ASCETデータベース／ワークスペース内でAUTOSARモデリングエレメント（ASCET V6.1 でサポートされているものは、モードグループ、インターフェース [1]、ソフトウェアコンポーネント）を作成し、管理します。

3.3 RTEジェネレータの使用法

AUTOSAR における開発フェーズと統合フェーズの分離は、以下の 2 フェーズからなるソフトウェアコンポーネント開発プロセスに反映されています。

1. ソフトウェアコンポーネントの開発：ソフトウェアコンポーネントの仕様決定・設計・実装を行います。
2. ソフトウェアコンポーネントの展開：ソフトウェアコンポーネントを ECU に割り当て、ECU 上の基本ソフトウェアと統合します。

[1] センダ／レシーバ、クライアント／サーバー、Calprm（AUTOSAR R3.1.5 以前）／Parameter（AUTOSAR R4.0.*）、NVData（AUTOSAR R4.0.*）

この 2 つのフェーズからなる作業を実行すると、ソフトウェアコンポーネントの初期のコンフィギュレーション設定を行った後、いくつかの補助的な設計・開発プロセスを経て VFB に統合し、その後、RTE インターフェースを生成することができるので、モデルのプロトタイプが作成されてソフトウェアコンポーネントが ECU 上に配置される前に、ソフトウェアコンポーネントを実装することができます。

このように開発プロセスがフェーズ分けされていると、「コンポーネント型」を開発してからそのコンポーネントプロトタイプを ECU に配置するまでに時間を置くことも可能になります。実際、一度開発されたコンポーネントは、その後、何世代もの自動車に再利用される可能性があります。しかもコンポーネントはインテグレータ（統合を担当する部署）に対してバイナリ形式でしか供給されない可能性がありますが、それをまだ作成されていない他のコンポーネントとともに ECU に統合しなければなりません。

RTE ジェネレータは、このようなフェーズ分けされたプロセスをサポートします。つまりコンポーネントプロトタイプから ECU への割り当てを完全に把握することにより、RTE へのインターフェースを前もって生成することを可能にします。RTE ジェネレータは、ソフトウェアコンポーネントディスクリプションを受け取り、その情報を用いてソフトウェアコンポーネント開発の開始に必要なインターフェース定義ファイルを生成します。インターフェースは、RTE とコンポーネントとの「コントラクト」、つまり以後の統合作業を容易にするためにコンポーネントが何を提供しなければならないか、という情報を定義します。このフェーズは「コントラクトフェーズ」と呼ばれます。

システムが統合され、ソフトウェアコンポーネントの ECU への割り当てがわかれば、RTE 自体の生成は可能になります。しかしその後、さまざまな補助的情報（ソフトウェアコンポーネントのインスタンス数、ランナブルエンティティが実行されるポイント、ECU の各通信についてのローカル／ネットワーク経由の区別など）が判明してくると、RTE ジェネレータは、それらの情報に基づく最適化を含んだインターフェース定義ファイルを再生成することができます。このフェーズは「RTE フェーズ」と呼ばれます。

以降の項では、コントラクトフェーズと RTE フェーズについてさらに詳しく説明します。

3.3.1 コントラクトフェーズ

コントラクトフェーズにおいて RTE ジェネレータは、コンポーネントが実装時に使用するヘッダファイルを作成します。ヘッダファイルは、コンポーネント（バイナリコードコンポーネントとソースコードコンポーネント）とシステムとの全体的な「コントラクト」（決まりごと）を定義するものです。コントラクトフェーズで実行する場合、RTE ジェネレータはソフトウェアコンポーネントディスクリプションファイルにだけアクセスできればよく、システムの展開に関する情報は必要ありません。

ARXML ファイル内の定義が API の定義に使用され、有効でないランナブルエンティティを宣言すると、コンポーネントのコンパイル時にエラーが発生します。

3.3.2 RTEフェーズ

RTE フェーズにおいて RTE ジェネレータを使用するには、相当量のシステムエンジニアリングが必要です。AUTOSAR 開発プロセスでは、システムエンジニアリングプロセスに対する以下のようなさまざまな入力が想定されます。

- ソフトウェアコンポーネントディスクリプション ― ソフトウェアコンポーネントと、そのポート、内部ビヘイビア、実装特性、VFB への接続を想定した際にポートにより提供／要求されるインターフェース（コントラクトフェーズで使用されるものと同じ記述）
- ECU リソースディスクリプション ― ECU のハードウェア特性（通信ポートなど）
- システム制約ディスクリプション ― システムの各種側面（通信プロトコルなど）

AUTOSAR システム（つまりネットワーク経由で通信し合う複数の ECU にマッピングされる一連のソフトウェアコンポーネント）を構築するには、以下のものが定義されている必要があります。

- ECU コンフィギュレーションディスクリプション ― ソフトウェアコンポーネントの ECU へのマッピングや、ECU 上で使用可能なリソース
- システムコンフィギュレーションディスクリプション ― ネットワークトポロジなどの情報や、ECU 間通信の物理ネットワークへのマッピングなど

- ECU コンフィギュレーション ― エレメント間のマッピング（ランナブルエンティティから AUTOSAR オペレーティングシステムタスクへのマッピングや、AUTOSAR シグナルから AUTOSAR COM シグナルへのマッピングなど）

コンポーネントプロトタイプを ECU インスタンスに割り当てて AUTOSAR システムを設定した後は、「RTE 生成」フェーズにおいて RTE ジェネレータを使用して以下を作成します。

1. RTE 自体の実装コード
2. ユーザーのコンフィギュレーションから得られたマッピング情報により最適化されたコンポーネントヘッダファイル
3. ユーザーのランナブルエンティティがまとめられたオペレーティングシステムタスク
4. （必要に応じて）生成された RTE オブジェクトと要求されたビヘイビアのためのオペレーティングシステムコンフィギュレーションファイル
5. （必要に応じて）ECU 間通信設定用の通信階層コンフィギュレーションファイル

RTE フェーズにおいて RTE は、ソースコードコンポーネントのコンパイルに適した最適化済みアプリケーションヘッダファイルを生成し、さらに必要に応じて、通信階層およびオペレーティングシステムのための XML コンフィギュレーションファイルも生成します。RTE フェーズ実行時には、RTE ジェネレータはすべてのシステム展開情報にアクセスする必要があります。

RTE は 1 つまたは複数の C モジュールとして生成され、各モジュールは RTE により出力される依存関係情報に従ってコンパイルされる必要があります。モジュール `Rte.c` には生成された RTE コアが含まれています。

4 データ型（AUTOSAR R3.1.5 以前）

インターフェース経由の通信は「型」により分類されているので、インターフェースを定義する前に、使用できるデータの型を定義しておく必要があります。

ASCET は基本型と複合型（複数の基本型の値で構成される型）の両方をサポートしています。

AUTOSAR の基本データ型と複合データ型の定義は、データ実装情報に基づいて ASCET が作成します。続いて、ASCET の実装データが AUTOSAR RTE により BSW 型にマッピングされます。

4.1 BSW型

AUTOSAR R3.1.5 以前のバージョンでは、AUTOSAR RTE は以下の BSW データ型をサポートしています。

- sint8 – 8 ビット符号付き整数
- uint8 – 8 ビット符号なし整数
- sint16 – 16 ビット符号付き整数
- uint16 – 16 ビット符号なし整数
- sint32 – 32 ビット符号付き整数
- uint32 – 32 ビット符号なし整数
- float32 – 単精度浮動小数点数
- float64 – 倍精度浮動小数点数
- uint8_least – 8 ビット以上の符号なし整数
- uint16_least – 16 ビット以上の符号なし整数
- uint32_least – 32 ビット以上の符号なし整数
- sint8_least – 8 ビット以上の符号付き整数
- sint16_least – 16 ビット以上の符号付き整数
- sint32_least – 32 ビット以上の符号付き整数
- boolean – TRUE/FALSE の 2 値をとる型

BSW 型、および TRUE と FALSE の定義は、RTA-RTE インストールディレクトリ内にある AUTOSAR ヘッダファイル `Platform_Types.h` に格納されます。

4.2 基本データ型

ASCET のデータ型システムはモデル型と実装型からなります。モデル型は 1 つまたは複数の実装型で実現できる抽象汎用型です。

スカラエレメント用の基本モデル型は以下のとおりです。

- Logic（論理）
- Signed Discrete（符号付き離散）
- Unsigned Discrete（符号なし離散）
- Continuous（連続）

ASCET のすべてのスカラエレメントは、以下の実装型のいずれかを使用して実装する必要があります。

- sint8
- sint16
- sint32
- uint8
- uint16
- uint32

モデル型 cont は以下の型でも実装できます。

- real64
- real32

モデル型 log は以下の型でも実装できます。

- bool

モデル型のデフォルトインプリメンテーションを設定する:

- コンポーネントマネージャで **Tools → Options** を選択します。
  “Options”ダイアログボックスが開きます。
- “Modeling/Implementation/default Implementation Types”ノードを開きます。
- デフォルトの実装型を下図の例のように設定します。

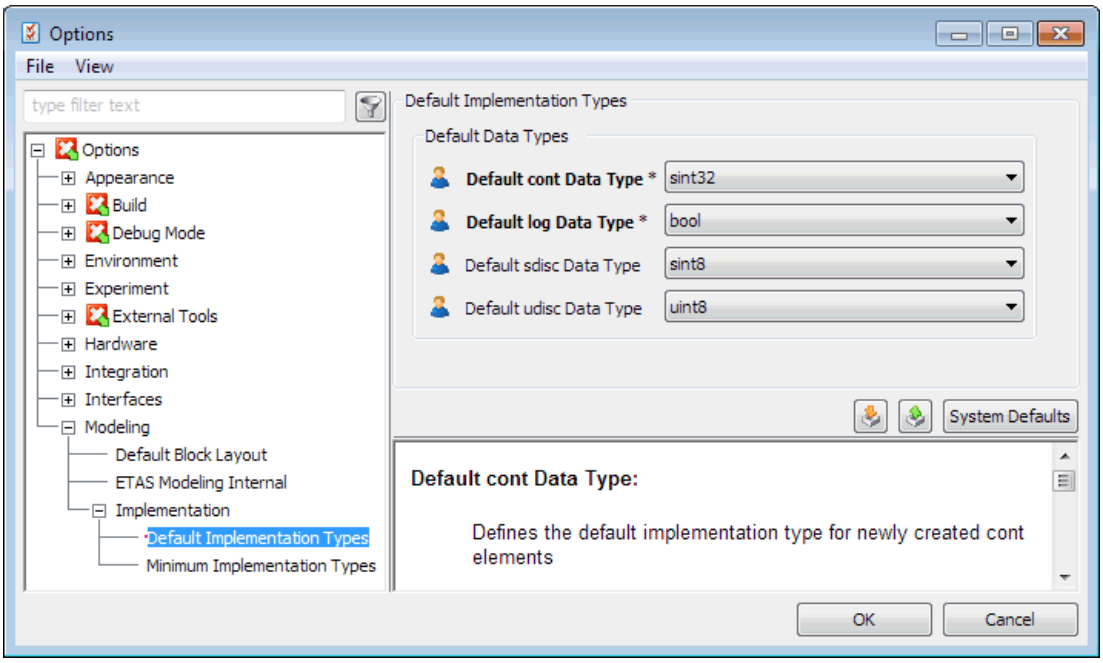


図 9: モデル型のデフォルトインプリメンテーション

- **OK** をクリックします。

モデルエレメントのインプリメンテーションは個別に設定できます。以下に、変数 sdisc を 8 ビット符号付き整数として実装する方法を説明します。

モデル型 sdisc を sint8 として実装する:

- コンポーネントマネージャから、3.1.1 項「AUTOSAR コンポーネントの作成に関する設定」で作成したプロジェクト ARProject を選択して **Edit → Open Component** を選択するか、またはこのプロジェクトをダブルクリックします。
  プロジェクトエディタウィンドウが開きます。
- プロジェクトエディタで、ソフトウェアコンポーネント Swc をダブルクリックします。
  ソフトウェアコンポーネントエディタウィンドウが開きます。
- **Signed Discrete Variable** ボタンをクリックして sdisc 変数を作成します。
  “Properties for Scalar Element: sdisc”ダイアログボックスが開きます。

- この変数の名前 sdisc（デフォルト名）をそのまま使用するので、**OK** をクリックしてプロパティダイアログボックスを閉じます。
- “Outline”タブで sdisc エレメントを右クリックし、ショートカットメニューから **Implementation** を選択します。
  “Implementation for: sdisc”ダイアログボックスが開きます。
- “Master”フィールドで、**Implementation** をオンにします。
- “Implementation”フィールドで、sint8 を選択します。
- **OK** をクリックして“Implementation for: sdisc”ダイアログボックスを閉じます。

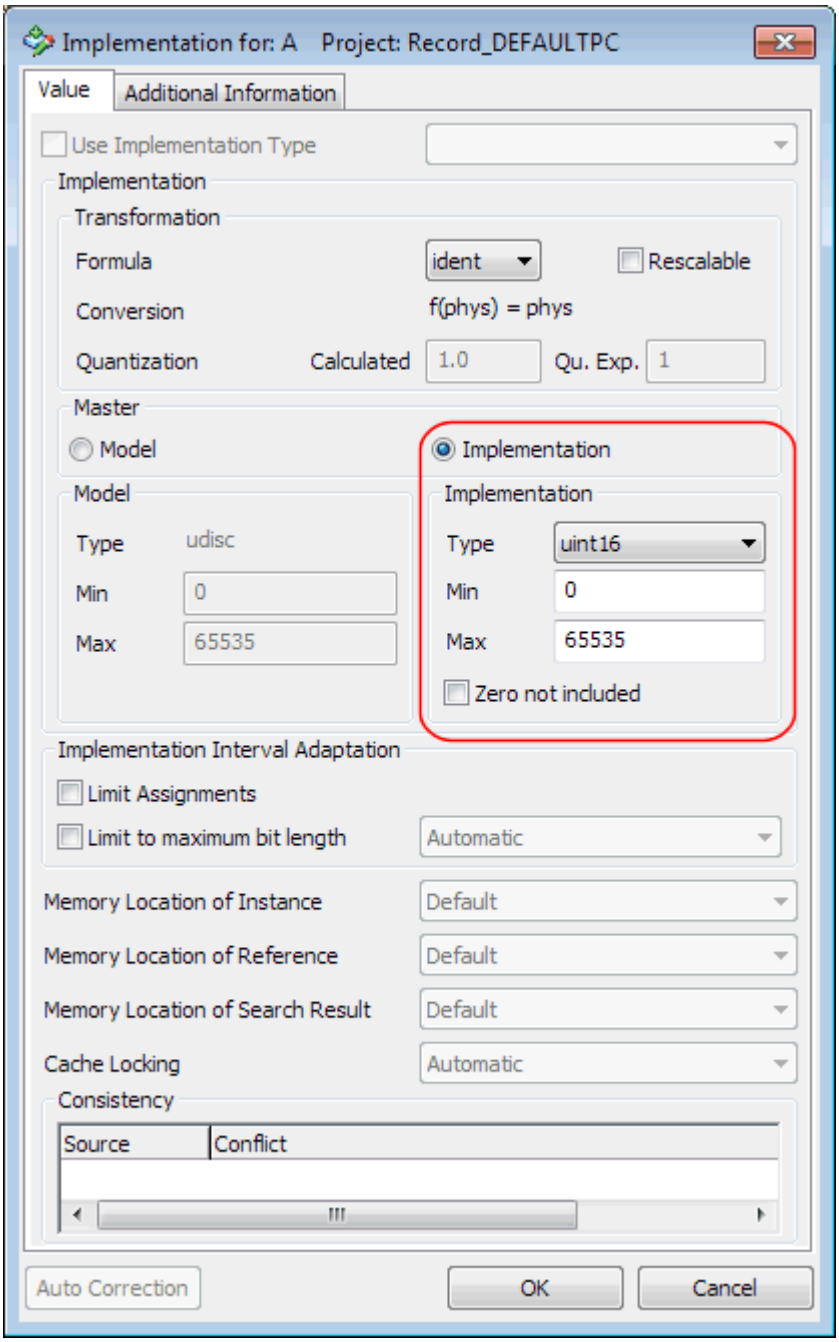


図 10: 符号付き離散エレメント sdisc を sint8 に変換するように設定されたインプリメンテーション

ASCET は、AUTOSAR プロジェクト用のコードを生成する際、各 ARXML エレメント（インターフェースなど）の中で基本データ型を参照できるようにするため、autosar_types.arxml というファイル

を作成します。基本型は、その型のプロパティを定義するメタタイプタグと、その型の範囲と型名を設定するサブタグを使用して以下のように宣言されます。

```
<AR-PACKAGE>
   <SHORT-NAME>AUTOSAR_types</SHORT-NAME>
   <DESC></DESC>
   <ELEMENTS>
      ...
      <INTEGER-TYPE>
         <SHORT-NAME>SInt8</SHORT-NAME>
         <LOWER-LIMIT>-128</LOWER-LIMIT>
         <UPPER-LIMIT>127</UPPER-LIMIT>
      </INTEGER-TYPE>
      ...
   </ELEMENTS>
</AR-PACKAGE>
```

コード 1: ARXML コード – 基本データ型（AUTOSAR R3.1.2）

データ型のショートネームは有効な C 識別子である必要があります。

型ファイル `autosar_types.arxml` は RTE ジェネレータへの入力です。ユーザー定義された基本型の型定義は、生成されるファイル `Rte_Type.h` に含まれます。RTE が BSW データ型を参照して基本型から作成する実装型は、特定のマイクロコントローラターゲット用 AUTOSAR ヘッダファイル `Platform_Types.h` に定義されます。

4.3 セマンティクス付きの基本データ型

ASCET では「列挙型」というデータ型を使用できます。

ASCET における列挙型は、セマンティックス付きの整数型に相当します。セマンティックスはカテゴリ Text Table の compu-method により与えられます。compu-method は、ビットパターンから物理値へ、およびその逆の変換式です。

列挙型を作成する:

- コンポーネントマネージャで **Insert → Enumeration** を選択するか、または **Enumeration** ボタンをクリックします。
- この列挙型の名前を `Enumeration` にします。
  “Contents”ペインに列挙型 `Enumeration` の内容が表示されます。
- 値 0 について、**Enumeration → Rename** を選択してラベル `red` を設定します。
- **Enumeration → Add Enumeration → Append** を選択するか、または **<INSERT>** キーを押して、新しい列挙子作成し、その値を 1 にしてラベルを `yellow` に変更します。
- もう 1 つ列挙子を追加し、値を 2、ラベルを `green` にします。

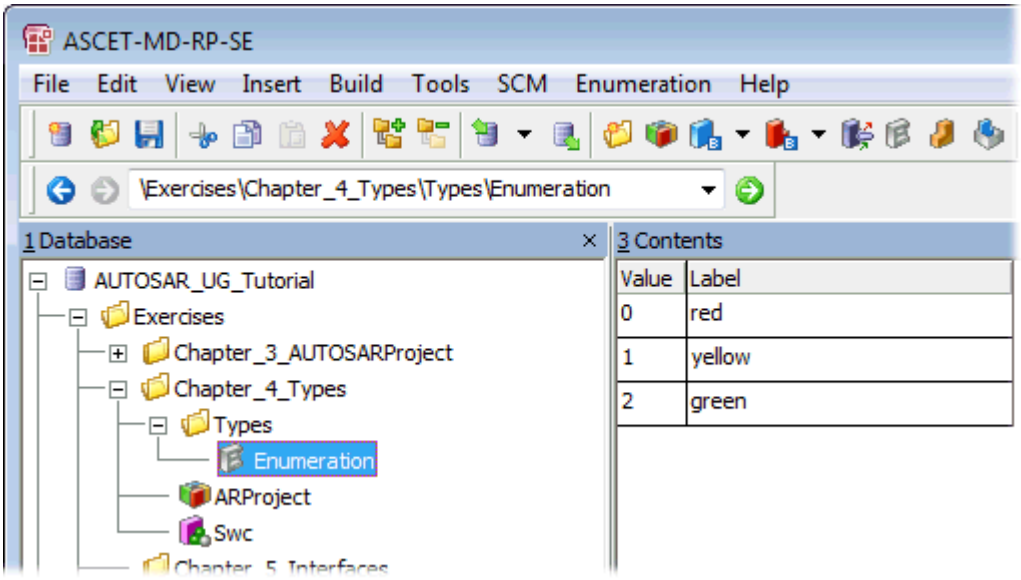


図 11: ASCET における列挙型の例

データ型と compu-method は、AUTOSAR パッケージ ASCET_types 内にコンフィギュレーション言語で以下のように定義されます。このパッケージ ASCET_types にはソフトウェアコンポーネント固有の型が含まれます。このパッケージはソフトウェアコンポーネントの型ファイル（生成されるファイル Swc_Types.arxml）に格納されます。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_types</SHORT-NAME>
  <DESC></DESC>
  <ELEMENTS>
    <INTEGER-TYPE>
      <SHORT-NAME>Enumeration</SHORT-NAME>
      <DESC></DESC>
      <!--
        enumeration &quot;Enumeration&quot; { red, yellow, green }
      -->
      <SW-DATA-DEF-PROPS>
        <COMPU-METHOD-REF DEST="COMPU-METHOD">
        /ASCET_types/enumerations/Enumeration</COMPU-METHOD-REF>
      </SW-DATA-DEF-PROPS>
      <LOWER-LIMIT>0</LOWER-LIMIT>
      <UPPER-LIMIT>2</UPPER-LIMIT>
    </INTEGER-TYPE>
    ...
  </ELEMENTS>
  <SUB-PACKAGES>
  </SUB-PACKAGES>
</AR-PACKAGE>
```

コード 2: ARXML コード – 列挙型（AUTOSAR R3.1.2）

compu-method のディスクリプションは以下のとおりです。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_types</SHORT-NAME>
  <DESC></DESC>
  <ELEMENTS>
  <SUB-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>enumerations</SHORT-NAME>
      <DESC></DESC>
      <ELEMENTS>
        <COMPU-METHOD>
          <SHORT-NAME>Enumeration</SHORT-NAME>
          ...
          <CATEGORY>TEXTTABLE</CATEGORY>
          <COMPU-INTERNAL-TO-PHYS>
            <COMPU-SCALES>
              <COMPU-SCALE>
                <LOWER-LIMIT>0</LOWER-LIMIT>
                <UPPER-LIMIT>0</UPPER-LIMIT>
                <COMPU-CONST>
                  <VT>red</VT>
                </COMPU-CONST>
              </COMPU-SCALE>
              <COMPU-SCALE>
                <LOWER-LIMIT>1</LOWER-LIMIT>
                <UPPER-LIMIT>1</UPPER-LIMIT>
                <COMPU-CONST>
                  <VT>yellow</VT>
                </COMPU-CONST>
              </COMPU-SCALE>
              <COMPU-SCALE>
                <LOWER-LIMIT>2</LOWER-LIMIT>
                <UPPER-LIMIT>2</UPPER-LIMIT>
                <COMPU-CONST>
                  <VT>green</VT>
                </COMPU-CONST>
              </COMPU-SCALE>
            </COMPU-SCALES>
          </COMPU-INTERNAL-TO-PHYS>
        </COMPU-METHOD>
      </ELEMENTS>
    </AR-PACKAGE>
    ...
  </SUB-PACKAGES>
</AR-PACKAGE>
```

コード 3: ARXML コード – 列挙型用 compu-method（AUTOSAR R3.1.2）

4.3.1 Std_ReturnType

AUTOSAR 規格では、RTE API 関数から返される ステータスとエラーの値が定義されています。値は以下のとおりで、Std_ReturnType 型で定義されています。

エラーコード	コードがサポートされている AUTOSAR バージョン		
	R4.0.*	R3.*	R2.*
RTE_E_COM_STOPPED	✓	✓	
RTE_E_COMMS_ERROR			✓
RTE_E_INVALID	✓	✓	✓
RTE_E_LIMIT	✓	✓	✓
RTE_E_LOST_DATA	✓	✓	✓
RTE_E_MAXAGE_EXCEEDED	✓	✓	✓
RTE_E_NO_DATA	✓	✓	✓
RTE_E_OK	✓	✓	✓
RTE_E_TIMEOUT	✓	✓	✓
RTE_E_TRANSMIT_ACK	✓	✓	X

表 2: AUTOSAR エラーコード

ASCET は Std_ReturnType 型をビルトイン列挙型として提供しています。これらのエラーコードは ASCET では予約語になっているので、他の列挙型で使用することはできません。

また、E_OK も ASCET で予約されている語です。これは、サーバーランナブルがアプリケーションエラーを返していないことを意味します。ユーザーはアプリケーションエラーとして発生する可能性のある値を標準の列挙型内に宣言、またはインポートする必要があります。

4.4 複合型

4.4.1 レコード型

「レコード型」を使用して新しい複合型を作成することができます。レコード型は 1 つまたは複数の名前付きメンバからなるデータ構造を定義するものです。

ASCET でレコードを作成する:

- コンポーネントマネージャで **Insert → Record** を選択するか、または **Record** ボタンをクリックします。
- そのレコードの名前を Record にします。
- **Edit → Open Component** を選択するか、そのレコードをダブルクリックします。

  レコードエディタが開きます。

- **Unsigned Discrete Variable** ボタンをクリックして変数 udisc を作成します。

  “Properties for Scalar Element: udisc”ダイアログボックスが開きます。
- 変数名を A に変更します。

- **Logic Variable** で、B という名前の log 変数を作成します。

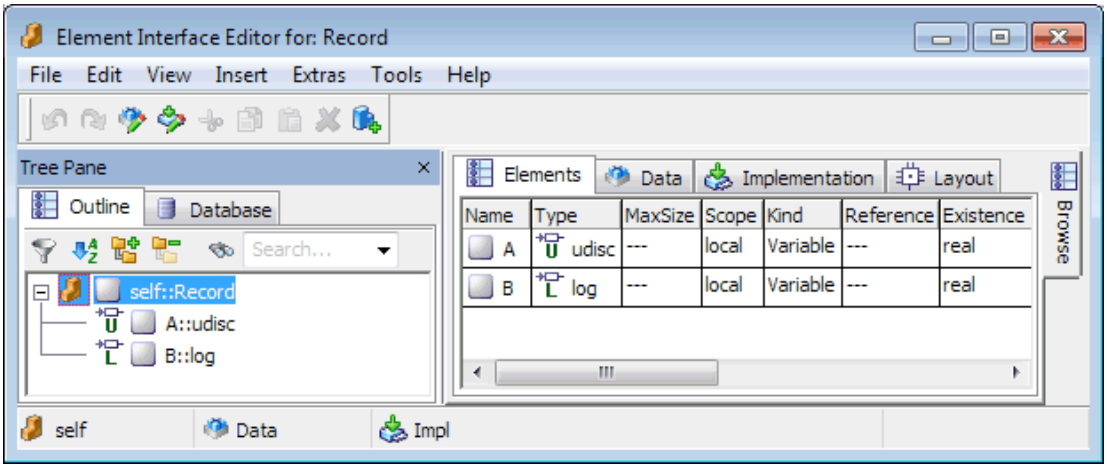


図 12: エレメント A および B を持つレコード

レコードのインプリメンテーションを設定する:

- レコードエディタの“Elements”タブから“Implementation”タブに切り替えます。
- “Implementation”タブ上のエレメント A をダブルクリックします。
  “Implementation for: A”ダイアログボックスが開きます。
- “Master”フィールドで **Implementation** をオンにします。
- “Implementation”フィールドで `uint16` を選択します。
- “Max”フィールドを右クリックしてショートカットメニューを開き、**Default Value** を選択します。
- **OK** をクリックして“Implementation for: A”ダイアログボックスを閉じます。

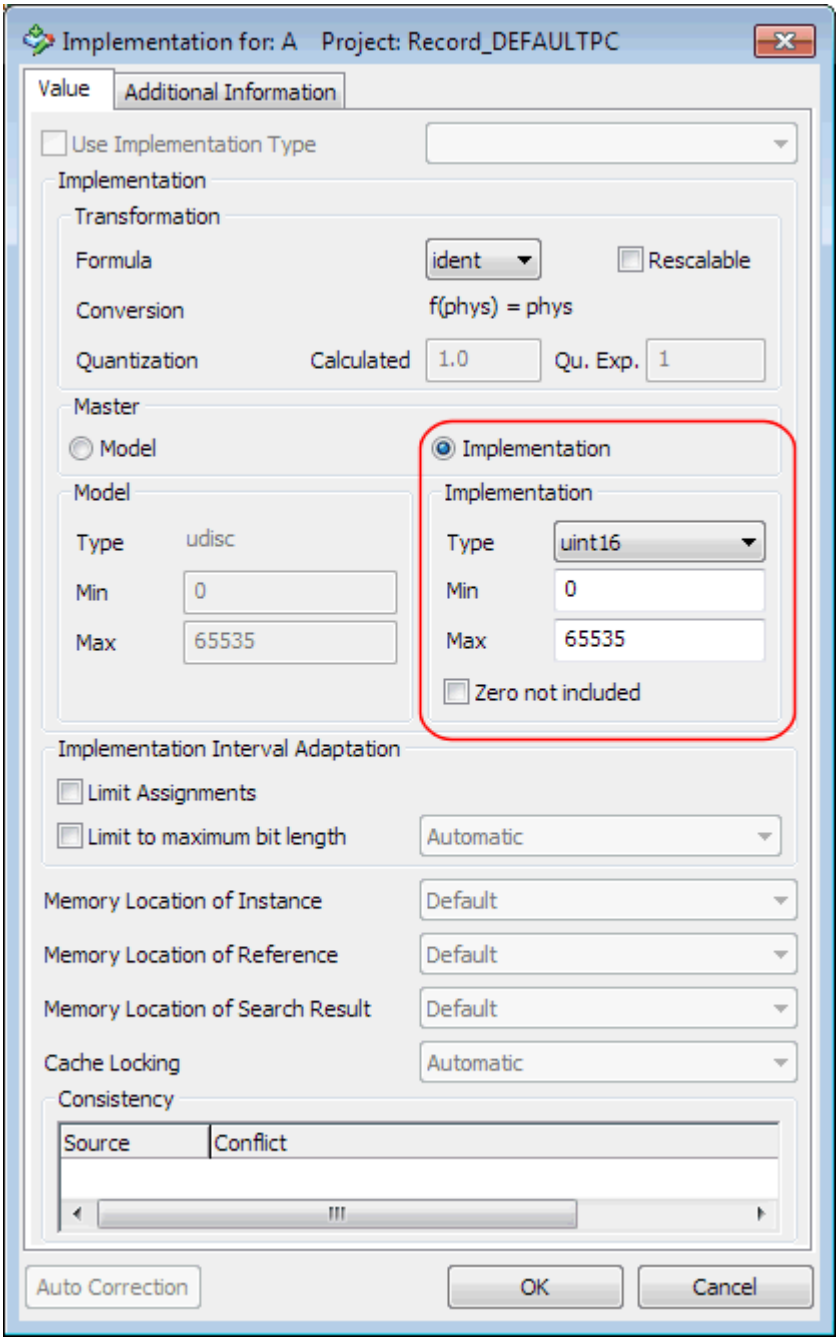


図 13: 符号なし離散エレメント A を `uint16` で実装するように設定されたインプリメンテーション

- 論理変数 B について、実装型 `bool` を選択します。
  レコードエディタの“Implementation”タブは以下のようになります。

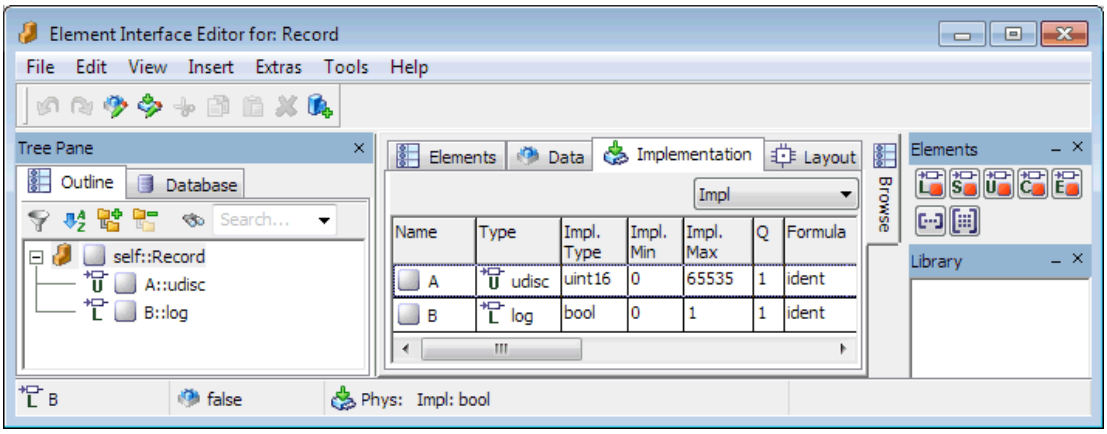


図 14: エレメント A および B を持つ Record のインプリメンテーション Impl

ASCET のレコードを実装したものは AUTOSAR のレコード型に相当します。レコード型は AUTOSAR パッケージ ASCET_types 内にコンフィギュレーション言語で定義されます。パッケージ ASCET_types にはソフトウェアコンポーネント固有の型が定義され、このパッケージはソフトウェアコンポーネントの型ファイル（生成されるファイル Swc_Types.arxml）に格納されます。レコード型 Record_Impl のメンバは以下のように記述されます。

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_types</SHORT-NAME>
   <DESC></DESC>
   <ELEMENTS>
      ...
      <RECORD-TYPE>
         <SHORT-NAME>Record_Impl</SHORT-NAME>
         <ELEMENTS>
            <RECORD-ELEMENT>
               <SHORT-NAME>A</SHORT-NAME>
               <TYPE-TREF DEST="INTEGER-TYPE">/AUTOSAR_types/UInt16</TYPE-TREF>
            </RECORD-ELEMENT>
            <RECORD-ELEMENT>
               <SHORT-NAME>B</SHORT-NAME>
               <TYPE-TREF DEST="BOOLEAN-TYPE">/AUTOSAR_types/Boolean</TYPE-TREF>
            </RECORD-ELEMENT>
         </ELEMENTS>
      </RECORD-TYPE>
   </ELEMENTS>
   ...
</AR-PACKAGE>
```

コード 4: ARXML コード – レコード型（AUTOSAR R3.1.2）

RTE ジェネレータは定義された<RECORD-TYPE>ごとに C 構造型を 1 つずつ生成します。構造の定義は生成されるファイル Rte_Type.h に含まれます。

レコードの新しいインプリメンテーションを作成する:

- レコードエディタで、**Edit → Component → Implementation** を選択します。
  “Implementation Editor for: Record”ダイアログボックスが開きます。
- **Implementation → Add** を選択し、そのインプリメンテーションの名前を指定します（例: Impl32）。
- A にインプリメンテーション uint32 を設定します。
- B にインプリメンテーション bool を設定します。

- **OK** をクリックします。

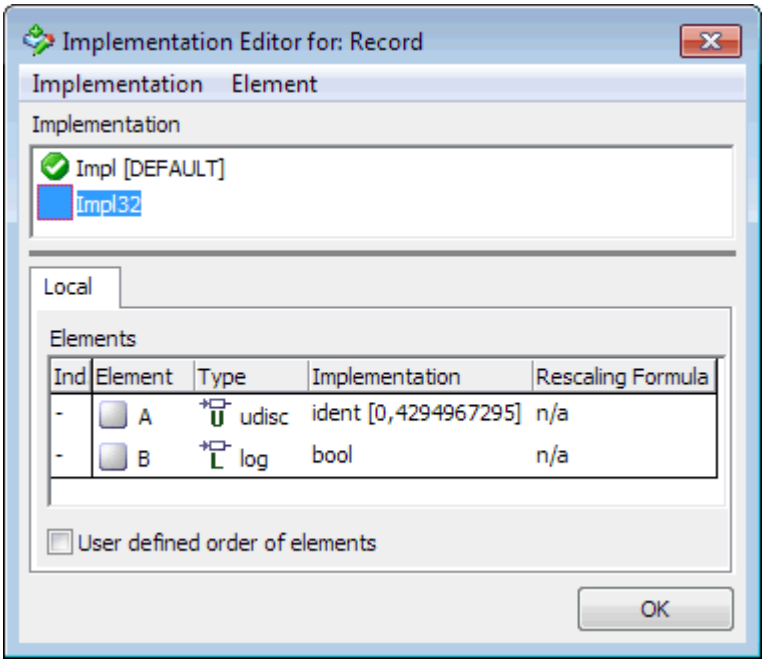


図 15: レコード型 `Record_Impl32`

4.4.2 配列型

「配列型」の場合もレコード型と同様、新しい複合型を作成することができます。配列型は、各インデックス位置にマッピングされる一連の値を作成するものです。

配列を作成する:

- コンポーネントマネージャで、プロジェクト `ARProject` を選択し、**Edit → Open Component** を選択します。
  プロジェクトエディタが開きます。
- プロジェクトエディタで、ソフトウェアコンポーネント `Swc` をダブルクリックします。
  ソフトウェアコンポーネントエディタが開きます。
- **Array** ボタンを使用して配列を作成します。
  配列用プロパティエディタが開きます。
- 変数名を `array` にして、X の要素数を `16` にセットし、さらに基本型 `unsigned discrete` を選択します。
- **OK** をクリックしてプロパティエディタを閉じます。
- SWC エディタの“Outline”タブで、配列を右クリックしてショートカットメニューから **Implementation** を選択します。
  “Implementation for: array”ダイアログボックスが開きます。
- “Master”フィールドで **Implementation** をオンにします。
- “Implementation”フィールドで `uint8` を選択します。
- **OK** をクリックして“Implementation for: array”ダイアログボックスを閉じます。

ASCET の配列を実装したものは AUTOSAR の配列型に相当します。配列型は AUTOSAR パッケージ `ASCET_types` 内にコンフィギュレーション言語で定義されます。パッケージ `ASCET_types` にはソフトウェアコンポーネント固有の型が定義され、このパッケージはソフトウェアコンポーネントの型ファイル（生成されるファイル `Swc_Types.arxml`）に格納されます。インターフェースエレメントなどに使用される配列型 `RASCET_Array_Uint8_16` は以下のように記述されます。

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_types</SHORT-NAME>
   <DESC></DESC>
   <ELEMENTS>
      ...
      <ARRAY-TYPE>
         <SHORT-NAME>ASCET_Array_UInt8_16</SHORT-NAME>
         <DESC></DESC>
         <!--
            array of 16 &quot;UInt8&quot; values
         -->
         <ELEMENT>
            <SHORT-NAME>elementName</SHORT-NAME>
            <TYPE-TREF DEST="INTEGER-TYPE">/AUTOSAR_types/UInt8</TYPE-TREF>
            <MAX-NUMBER-OF-ELEMENTS>16</MAX-NUMBER-OF-ELEMENTS>
         </ELEMENT>
      </ARRAY-TYPE>
      ...
   </ELEMENTS>
   ...
</AR-PACKAGE>
```

コード 5: ARXML コード – 配列型（AUTOSAR R3.1.2）

RTE ジェネレータは定義された<ARRAY-TYPE>ごとに C 配列型を 1 つずつ生成するので、配列型は C 配列と同じセマンティックスで宣言する必要があります。配列型の定義は生成されるファイル `Rte_Type.h` に含まれます。

注記

アプリケーションソフトウェアコンポーネント内の配列のインプリメンテーションは、生成される RTE 内の宣言と一致している必要があります。AUTOSAR R3.1.2 では、配列型エレメントを C コードレベルで宣言する方法について最初に規定されています。詳細については、使用している AUTOSAR バージョンのマニュアル `AUTOSAR_SWS_RTE.pdf` の 5.2.6.5 項を参照してください。

ASCET における配列の C コード生成については、ファイル `codegen.ini` 内でオプション **ARArrayBaseTypePassing** により設定できます

5 データ型（AUTOSAR R4.0.*）

AUTOSAR R4.0.*のデータ型のメタモデルは、それまでのデータ型を全面的に変更したものとなっています。AUTOSAR R4.0.*におけるデータ型は、図 16.に示されるように 3 つのレベルに抽象化されています。

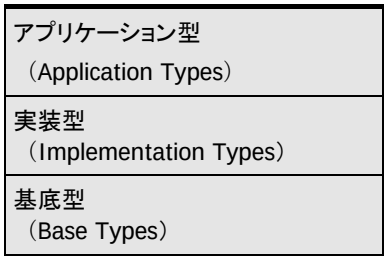


図 16: AUTOSAR R4.0.* におけるデータ型の抽象化レベル

5.1 アプリケーションデータ型（Application Data Types）

アプリケーションデータ型は物理単位で定義されます。これを用いることにより、アプリケーションの作成者はライフサイクルの初期において C データ型を決定する必要がなくなります。

アプリケーションデータ型には測定／適合ツールに対応するために必要な情報が含まれています。

またアプリケーションデータ型は値の単位の自動変換にも対応しています。

アプリケーションデータ型の<SHORT-NAME>がソフトウェアコンポーネント型（SWCT）として使用されるので、複数の SWCT を統合する際に、1 つの ECU 上で同じ名前のアプリケーションデータ型を複数持つことが可能です（ただし 1 つの SWCT 内ではこの限りではありません）。

アプリケーションデータ型の<SHORT-NAME>は、生成されたコード内では使用されません。つまり RTE の API はマッピングされた実装データ型として定義されます。

より複雑なデータ型を使用する必要がある場合は、アプリケーションデータ型から別のアプリケーション型を構成することができます。このような再帰的な定義により、レコードと配列の定義が可能です。

RTE の生成時には、使用されるアプリケーションデータ型が実装型にマッピングされる必要があります。詳しくは 5.3 項「型のマッピング」を参照してください。

5.2 実装データ型（Implementation Data Types）

実装データ型は、生成されるコード内の C データ型を表します。実装データ型の<SHORT-NAME>が、C コード（API やユーザーコードなど）でデータ型を表すシンボルを定義します。

一般的に実装データ型は、生成される C コード内の `typedef` 文となります。例外については RTA-RTE ユーザーズガイドを参照してください。

RTA-RTE は、生成される API 内で常に実装データ型を使用します。対応する<Variable-Data-Prototype>がアプリケーションデータ型への参照として定義されている場合は、マッピングされる実装データ型が API シグネチャ内で使用されます。

5.3 型のマッピング

アプリケーションデータ型（5.1 項参照）から実装データ型（5.2 項参照）へのマッピングには、SWC 固有の「データ型マッピング」が使用されます。

モード宣言グループから実装データ型へのマッピングには「モード型マッピング」が使用されます。

注記

RTA-RTE では、RTE 用コードを生成する際に、各アプリケーションデータ型用のデータ型マッピング、および使用される各モード宣言グループ用のモード型マッピングが必要です。

ASCET では、これらのマッピングは Swc_mappings.arxml というファイル内に提供されます。

SWC 用のデータ型マッピングは、<DATA-TYPE-MAPPING-SET>エレメント内に定義されます。

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
   <AR-PACKAGES>
      <AR-PACKAGE>
         <SHORT-NAME>DataMappings</SHORT-NAME>
         <ELEMENTS>
            <DATA-TYPE-MAPPING-SET>
               <SHORT-NAME>Impl</SHORT-NAME>
               <DATA-TYPE-MAPS>
                  <DATA-TYPE-MAP>
                     ...
                  </DATA-TYPE-MAP>
                  ...
               </DATA-TYPE-MAPS>
               <MODE-REQUEST-TYPE-MAPS>
                  <MODE-REQUEST-TYPE-MAP>
                     ...
                  </MODE-REQUEST-TYPE-MAP>
               </MODE-REQUEST-TYPE-MAPS>
            </DATA-TYPE-MAPPING-SET>
         </ELEMENTS>
      </AR-PACKAGE>
   </AR-PACKAGES>
</AR-PACKAGE>
```

コード 6: ARXML コード ― アプリケーションデータ型とモード型から実装型へのマッピング（AUTOSAR R4.0.2）

データ型マッピングには 1 つ以上のデータ型マッピング情報が含まれています。各マッピング情報は 1 つのアプリケーションデータ型と 1 つの実装データ型を参照しています（ARMXL の例: コード 8、コード 12、コード 15、コード 21 を参照）。

データ型とモードマッピングについての詳しい情報は、RTA-RTE ユーザーズガイドを参照してください。

5.4 プラットフォーム型（Platform Types）

AUTOSAR では C コード内で使用する一連の「プラットフォーム型」が定義されています。これにより、異なるターゲットハードウェアに対して同じセマンティックスを持つ一連の型を提供することが可能になります。RTA-RTE は、内部変数用のデータ型を作成する必要が生じた際にプラットフォーム型を使用します。

一般的な実装データ型とは異なり、プラットフォーム型は C 言語の PlatformTypes.h ファイル内にも定義されます。

AUTOSAR R4.0.2 からは、プラットフォーム型の正確な定義とパッケージ名が決められています。

プラットフォーム型は、AUTOSAR Specification of Platform Types および標準のヘッダファイル Platform_Types.h 内に、以下のように定義されています。

- sint8 – 8 ビット符号付き整数
- uint8 – 8 ビット符号なし整数
- sint16 – 16 ビット符号付き整数
- uint16 – 16 ビット符号なし整数
- sint32 – 32 ビット符号付き整数
- uint32 – 32 ビット符号なし整数
- float32 – 単精度浮動小数点
- float64 – 倍精度浮動小数点
- uint8_least – 8 ビット以上の符号なし整数
- uint16_least –16 ビット以上の符号なし整数
- uint32_least – 32 ビット以上の符号なし整数
- sint8_least – 7 ビット以上の符号付き整数（+符号ビット）
- sint16_least – 15 ビット以上の符号付き整数（+符号ビット）
- sint32_least – 31 ビット以上の符号付き整数（+符号ビット）
- boolean – TRUE/FALSE.

5.5 基底型（Base Types）

最も下位レベルの基底型は、ハードウェアのアーキテクチャ（サイズやエンコーディングなど）に対応するデータ型です。

これをもとに実装データ型が作成されます。各基底型は、複数の実装データ型（5.2 項参照）により参照されます。

基底型の<SHORT-NAME>は、生成されるコード内では使用されず、モデル内で「参照ターゲット」としてのみ使用されます。生成されるコード内で使用されるのは実装データ型のみです。

5.6 使用例

本項では、アプリケーションデータ型、実装データ型、プラットフォーム型、基底型の例を示します。ここでは第 4 章の「データ型（AUTOSAR R3.1.5 以前）」（33 ページ）で使用されているモデルが使用されています。

5.6.1 基本アプリケーションデータ型

AUTOSAR R4.0.*プロジェクトのコードを生成する際には、ASCET は Swc_appltypes.arxml および Swc_impltypes.arxml というファイルを生成し、AUTOSAR_MOD_PlatformTypes.arxml および AUTOSAR_MOD_PlatformBaseTypes_TC1796.arxml というファイルをコード生成ディレクトリにコピーします。

以下の例は、実装データ型が sint8 である変数 sdisc （33 ページの 4.2 項「基本データ型」を参照）について Swc_appltypes.arxml 内に定義された基本アプリケーションデータ型を示しています。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Types</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
      <ELEMENTS>
        ...
        <APPLICATION-PRIMITIVE-DATA-TYPE>
          <SHORT-NAME>SInt8</SHORT-NAME>
          <CATEGORY>VALUE</CATEGORY>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
                <DATA-CONSTR-REF DEST="DATA-CONSTR">
                /ASCET_DataConstrs/Physical/dc_SInt8</DATA-CONSTR-REF>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </APPLICATION-PRIMITIVE-DATA-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 7: ARXML コード – 基本アプリケーションデータ型 `SInt8`（AUTOSAR R4.0.2）

`Swc_mappings.arxml` ファイル内では、アプリケーションデータ型が実装データ型にマッピングされています。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>DataMappings</SHORT-NAME>
      <ELEMENTS>
        <DATA-TYPE-MAPPING-SET>
          <SHORT-NAME>Impl</SHORT-NAME>
          <DATA-TYPE-MAPS>
            ...
            <DATA-TYPE-MAP>
              <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
              /ASCET_Types/ApplicationDataTypes/SInt8</APPLICATION-DATA-TYPE-REF>
              <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
              /AUTOSAR_PlatformTypes/ImplementationDataTypes/sint8
              </IMPLEMENTATION-DATA-TYPE-REF>
            </DATA-TYPE-MAP>
            ...
          </DATA-TYPE-MAPS>
          <MODE-REQUEST-TYPE-MAPS>
          ...
          </MODE-REQUEST-TYPE-MAPS>
        </DATA-TYPE-MAPPING-SET>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 8: ARXML コード – アプリケーションデータ型 `SInt8` と実装データ型 `sint8` のマッピング（AUTOSAR R4.0.2）

参照される実装データ型はプラットフォーム型なので、Swc_impltypes.arxml ファイル内には含まれません。参照される実装データ型は、AUTOSAR_MOD_PlatformTypes.arxml ファイル内に以下のように定義されます。

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR_PlatformTypes</SHORT-NAME>
  ...
  <AR-PACKAGES>
    ...
    <AR-PACKAGE>
      <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">AUTOSAR Platform types</L-4>
      </LONG-NAME>
      <ELEMENTS>
        ...
        <IMPLEMENTATION-DATA-TYPE>
          <SHORT-NAME>sint8</SHORT-NAME>
          <LONG-NAME>
            <L-4 L="EN">signed integer 8bit</L-4>
          </LONG-NAME>
          <CATEGORY>VALUE</CATEGORY>
          <INTRODUCTION>
            <TRACE>
              <SHORT-NAME>PLATFORM016</SHORT-NAME>
              <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
              <P>
                <L-1 L="EN">This standard AUTOSAR type shall be 8 bit signed</L-1>
              </P>
            </TRACE>
          </INTRODUCTION>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <BASE-TYPE-REF DEST="SW-BASE-TYPE">
                /AUTOSAR_PlatformTypes/SwBaseTypes/sint8</BASE-TYPE-REF>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </IMPLEMENTATION-DATA-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
    ...
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 9: ARXML コード – プラットフォームデータ型 sint8（AUTOSAR R4.0.2）

参照される基底型は AUTOSAR_MOD_PlatformBaseTypes_TC1796.arxml ファイル内に以下のように定義されます。

```
<AR-PACKAGE>
  <SHORT-NAME>AUTOSAR_PlatformTypes</SHORT-NAME>
  <AR-PACKAGES>
    ...
    <AR-PACKAGE>
      <SHORT-NAME>SwBaseTypes</SHORT-NAME>
      <LONG-NAME>
        <L-4 L="EN">AUTOSAR Base Types for AUTOSAR Platform types for TC1796</L-4>
      </LONG-NAME>
      <ELEMENTS>
        ...
        <SW-BASE-TYPE>
          <SHORT-NAME>sint8</SHORT-NAME>
          <LONG-NAME>
            <L-4 L="EN">signed integer 8bit</L-4>
          </LONG-NAME>
          <CATEGORY>FIXED_LENGTH</CATEGORY>
          <INTRODUCTION>
            <TRACE>
              <SHORT-NAME>PLATFORM016</SHORT-NAME>
              <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
              <P>
                <L-1 L="EN">This standard AUTOSAR type shall be 8 bit signed</L-1>
              </P>
            </TRACE>
          </INTRODUCTION>
          <BASE-TYPE-SIZE>8</BASE-TYPE-SIZE>
          <BASE-TYPE-ENCODING>2C</BASE-TYPE-ENCODING>
        </SW-BASE-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 10: ARXML コード – 基底型 `sint8`（AUTOSAR R4.0.2）

5.6.2 列挙型（セマンティックス付きの基本データ型）

以下の例は、列挙型 `Enumeration`（36 ページの 4.3 項「セマンティクス付きの基本データ型」を参照）について `Swc_appltypes.arxml` 内に定義されたアプリケーションデータ型を示しています。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Types</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
      <ELEMENTS>
        ...
        <APPLICATION-PRIMITIVE-DATA-TYPE>
          <SHORT-NAME>Enumeration</SHORT-NAME>
          <CATEGORY>VALUE</CATEGORY>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
                <COMPU-METHOD-REF DEST="COMPU-METHOD">
                /ASCET_CompuMethods/Enumerations/Enumeration</COMPU-METHOD-REF>
                <DATA-CONSTR-REF DEST="DATA-CONSTR">
                /ASCET_DataConstrs/Physical/dc_m1to6</DATA-CONSTR-REF>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </APPLICATION-PRIMITIVE-DATA-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 11: ARXML コード – アプリケーションデータ型 `Enumeration`（AUTOSAR R4.0.2）

`Swc_mappings.arxml` ファイル内で、アプリケーションデータ型は以下のように実装データ型にマッピングされています。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>DataMappings</SHORT-NAME>
      <ELEMENTS>
        <DATA-TYPE-MAPPING-SET>
          <SHORT-NAME>Impl</SHORT-NAME>
          <DATA-TYPE-MAPS>
            ...
            <DATA-TYPE-MAP>
              <APPLICATION-DATA-TYPE-REF DEST=
              "APPLICATION-PRIMITIVE-DATA-TYPE">
              /ASCET_Types/ApplicationDataTypes/Enumeration
              </APPLICATION-DATA-TYPE-REF>
              <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
              /ASCET_Types/ImplementationDataTypes/Enumeration
              </IMPLEMENTATION-DATA-TYPE-REF>
            </DATA-TYPE-MAP>
            ...
          </DATA-TYPE-MAPS>
          <MODE-REQUEST-TYPE-MAPS></MODE-REQUEST-TYPE-MAPS>
        </DATA-TYPE-MAPPING-SET>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 12: ARXML コード – アプリケーションデータ型 `Enumeration` と実装データ型のマッピング（AUTOSAR R4.0.2）

参照される実装データ型はプラットフォーム型ではないため、Swc_impltypes.arxml ファイル内に含まれています。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Types</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
      <ELEMENTS>
        ...
        <IMPLEMENTATION-DATA-TYPE>
          <SHORT-NAME>Enumeration</SHORT-NAME>
          <CATEGORY>TYPE_REFERENCE</CATEGORY>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <COMPU-METHOD-REF DEST="COMPU-METHOD">
                /ASCET_CompuMethods/Enumerations/Enumeration</COMPU-METHOD-REF>
                <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
                /AUTOSAR_PlatformTypes/ImplementationDataTypes/sint8
                </IMPLEMENTATION-DATA-TYPE-REF>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
        </IMPLEMENTATION-DATA-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 13: ARXML コード – 実装データ型 Enumeration（AUTOSAR R4.0.2）

実装データ型はプラットフォーム型 sint8 を参照しています（49 ページのコード 9 を参照）。
プラットフォーム型 sint8 は基底型 sint8 を参照しています（50 ページのコード 10 を参照）。

5.6.3 レコード型（複合型）

レコード Record（4.4.1 項の「レコード型」を参照）については、Swc_appltypes.arxml ファイル内に以下のようなアプリケーションデータ型が定義されます。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Types</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
      <ELEMENTS>
        ...
        <APPLICATION-RECORD-DATA-TYPE>
          <SHORT-NAME>Record_Impl</SHORT-NAME>
          <CATEGORY>STRUCTURE</CATEGORY>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
          <ELEMENTS>
            <APPLICATION-RECORD-ELEMENT>
              <SHORT-NAME>A</SHORT-NAME>
              <CATEGORY>VALUE</CATEGORY>
              <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
              /ASCET_Types/ApplicationDataTypes/UInt16</TYPE-TREF>
            </APPLICATION-RECORD-ELEMENT>
            <APPLICATION-RECORD-ELEMENT>
              <SHORT-NAME>B</SHORT-NAME>
              <CATEGORY>VALUE</CATEGORY>
              <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
              /ASCET_Types/ApplicationDataTypes/Boolean</TYPE-TREF>
            </APPLICATION-RECORD-ELEMENT>
          </ELEMENTS>
        </APPLICATION-RECORD-DATA-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 14: ARXML コード – アプリケーションデータ型 `Record_Impl`（AUTOSAR R4.0.2）

`Swc_mappings.arxml` ファイル内で、アプリケーションデータ型 `Record_Impl` は以下のように実装データ型にマッピングされます。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>DataMappings</SHORT-NAME>
      <ELEMENTS>
        <DATA-TYPE-MAPPING-SET>
          <SHORT-NAME>Impl</SHORT-NAME>
          <DATA-TYPE-MAPS>
            ...
            <DATA-TYPE-MAP>
              <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-RECORD-DATA-TYPE">
              /ASCET_Types/ApplicationDataTypes/Record_Impl
              </APPLICATION-DATA-TYPE-REF>
              <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
              /ASCET_Types/ImplementationDataTypes/Record_Impl
              </IMPLEMENTATION-DATA-TYPE-REF>
            </DATA-TYPE-MAP>
          </DATA-TYPE-MAPS>
          <MODE-REQUEST-TYPE-MAPS></MODE-REQUEST-TYPE-MAPS>
        </DATA-TYPE-MAPPING-SET>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 15: ARXML コード – アプリケーションデータ型 `Record_Impl` と実装データ型のマッピング（AUTOSAR R4.0.2）

参照されている実装データ型はプラットフォーム型ではないため、`Swc_impltypes.arxml` ファイル内に含まれています。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Types</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
      <ELEMENTS>
        ...
        <IMPLEMENTATION-DATA-TYPE>
          <SHORT-NAME>Record_Impl</SHORT-NAME>
          <CATEGORY>STRUCTURE</CATEGORY>
          <SUB-ELEMENTS>
            <IMPLEMENTATION-DATA-TYPE-ELEMENT>
              <SHORT-NAME>A</SHORT-NAME>
              <CATEGORY>TYPE_REFERENCE</CATEGORY>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE"
                    >/AUTOSAR_PlatformTypes/ImplementationDataTypes/uint16
                    </IMPLEMENTATION-DATA-TYPE-REF>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
            </IMPLEMENTATION-DATA-TYPE-ELEMENT>
            <IMPLEMENTATION-DATA-TYPE-ELEMENT>
              <SHORT-NAME>B</SHORT-NAME>
              <CATEGORY>TYPE_REFERENCE</CATEGORY>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE"
                    >/AUTOSAR_PlatformTypes/ImplementationDataTypes/boolean
                    </IMPLEMENTATION-DATA-TYPE-REF>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
            </IMPLEMENTATION-DATA-TYPE-ELEMENT>
          </SUB-ELEMENTS>
        </IMPLEMENTATION-DATA-TYPE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 16: ARXML コード – 実装データ型 `Record_Impl`（AUTOSAR R4.0.2）

実装データ型 `Record_impl` は、2 つのプラットフォーム型（各レコードエレメントがそれぞれ1つずつ）を参照しています。参照される実装データ型は、`AUTOSAR_MOD_PlatformTypes.arxml` ファイル内に以下のように示されます。

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>boolean</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Boolean</L-4>
  </LONG-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <INTRODUCTION>
    <TRACE>
      <SHORT-NAME>PLATFORM026</SHORT-NAME>
      <CATEGORY>CONSTRAINT</CATEGORY>
      <P>
        <L-1 L="EN">This standard AUTOSAR type shall only be used together with the definitions
        TRUE and FALSE. See <XREF><REFERRABLE-REF DEST="TRACEABLE-TEXT" BASE="SWS_PlatformTypes"
        >PLATFORM027</REFERRABLE-REF></XREF> for implementation and usage.</L-1>
      </P>
    </TRACE>
    <TRACE>
      <SHORT-NAME>PLATFORM060</SHORT-NAME>
      <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
      <P>
        <L-1 L="EN">The boolean type shall always be mapped to a platform specific type where
        pointers can be applied to to enable a passing of parameters via API. There are
        specific BIT types of some HW platforms which are very efficient but where no pointers
        can point to.</L-1>
      </P>
      <P>
        <L-1 L="EN">There are specific BIT types of some HW platforms which are very efficient
        but where no pointers can point to.</L-1>
      </P>
    </TRACE>
  </INTRODUCTION>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <!-- CompuMethod for TRUE and FALSE -->
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_PlatformTypes/SwBaseTypes/boolean
        </BASE-TYPE-REF>
        <COMPU-METHOD-REF DEST="COMPU-METHOD">/AUTOSAR_PlatformTypes/CompuMethods/boolean
        </COMPU-METHOD-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</IMPLEMENTATION-DATA-TYPE>
```

コード 17: ARXML コード – プラットフォームデータ型 Boolean（AUTOSAR R4.0.2）

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint16</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">unsigned integer 16bit</L-4>
  </LONG-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <INTRODUCTION>
    <TRACE>
      <SHORT-NAME>PLATFORM014</SHORT-NAME>
      <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
      <P>
        <L-1 L="EN">This standard AUTOSAR type shall be of 16 bit unsigned.</L-1>
      </P>
    </TRACE>
  </INTRODUCTION>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">/AUTOSAR_PlatformTypes/SwBaseTypes/uint16
        </BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</IMPLEMENTATION-DATA-TYPE>
```

コード 18: ARXML コード – プラットフォームデータ型 `uint16`（AUTOSAR R4.0.2）

コード 17 とコード 18 で参照されている基底型 `boolean` および `uint16` は、`AUTOSAR_MOD_PlatformBaseTypes_TC1796.arxml` ファイル内に以下のように定義されます。

```
<SW-BASE-TYPE>
  <SHORT-NAME>boolean</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">Boolean</L-4>
  </LONG-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <INTRODUCTION>
    <TRACE>
      <SHORT-NAME>PLATFORM060</SHORT-NAME>
      <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
      <P>
        <L-1 L="EN">The boolean type shall always be mapped to a platform specific type
        where pointers can be applied to to enable a passing of parameters via API. There
        are specific BIT types of some HW platforms which are very efficient but where
        no pointers can point to.</L-1>
      </P>
    </TRACE>
    <TRACE>
      <SHORT-NAME>PLATFORM026</SHORT-NAME>
      <P>
        <L-1 L="EN">This standard AUTOSAR type shall only be used together with the
        definitions TRUE and FALSE. See PLATFORM027 for implementation and usage.</L-1>
      </P>
    </TRACE>
  </INTRODUCTION>
  <BASE-TYPE-SIZE>8</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>BOOLEAN</BASE-TYPE-ENCODING>
</SW-BASE-TYPE>
...
<SW-BASE-TYPE>
  <SHORT-NAME>uint16</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">unsigned integer 16bit</L-4>
  </LONG-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <INTRODUCTION>
    <TRACE>
      <SHORT-NAME>PLATFORM014</SHORT-NAME>
      <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
      <P>
        <L-1 L="EN">This standard AUTOSAR type shall be of 16 bit unsigned.</L-1>
      </P>
    </TRACE>
  </INTRODUCTION>
  <BASE-TYPE-SIZE>16</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
</SW-BASE-TYPE>
```

コード 19: ARXML コード – 基底型 `boolean` および `uint16`（AUTOSAR R4.0.2）

5.6.4 配列型（複合型）

配列 `array`（4.4.2 項の「配列型」を参照）については、`Swc_appltypes.arxml` ファイル内に以下のようなアプリケーションデータ型が定義されます。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Types</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
      <ELEMENTS>
        <APPLICATION-ARRAY-DATA-TYPE>
          <SHORT-NAME>UInt8_16</SHORT-NAME>
          <!--
            array of 16 &quot;UInt8&quot; values
          -->
          <CATEGORY>ARRAY</CATEGORY>
          <SW-DATA-DEF-PROPS>
            <SW-DATA-DEF-PROPS-VARIANTS>
              <SW-DATA-DEF-PROPS-CONDITIONAL>
                <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
              </SW-DATA-DEF-PROPS-CONDITIONAL>
            </SW-DATA-DEF-PROPS-VARIANTS>
          </SW-DATA-DEF-PROPS>
          <ELEMENT>
            <SHORT-NAME>elementName</SHORT-NAME>
            <CATEGORY>VALUE</CATEGORY>
            <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
            /ASCET_Types/ApplicationDataTypes/UInt8</TYPE-TREF>
            <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
            <MAX-NUMBER-OF-ELEMENTS>16</MAX-NUMBER-OF-ELEMENTS>
          </ELEMENT>
        </APPLICATION-ARRAY-DATA-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 20: ARXML コード – カテゴリ ARRAY のアプリケーションデータ型 `UInt8_16`（AUTOSAR R4.0.2）

`Swc_mappings.arxml` ファイル内で、以下のようにアプリケーションデータ型が実装データ型にマッピングされています。

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_Mappings</SHORT-NAME>
   <AR-PACKAGES>
      <AR-PACKAGE>
         <SHORT-NAME>DataMappings</SHORT-NAME>
         <ELEMENTS>
            <DATA-TYPE-MAPPING-SET>
               <SHORT-NAME>Impl</SHORT-NAME>
               <DATA-TYPE-MAPS>
                  ...
                  <DATA-TYPE-MAP>
                     <APPLICATION-DATA-TYPE-REF DEST="APPLICATION-ARRAY-DATA-TYPE">
                     /ASCET_Types/ApplicationDataTypes/UInt8_16
                     </APPLICATION-DATA-TYPE-REF>
                     <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
                     /ASCET_Types/ImplementationDataTypes/uint8_16
                     </IMPLEMENTATION-DATA-TYPE-REF>
                  </DATA-TYPE-MAP>
                  ...
               </DATA-TYPE-MAPS>
               <MODE-REQUEST-TYPE-MAPS></MODE-REQUEST-TYPE-MAPS>
            </DATA-TYPE-MAPPING-SET>
         </ELEMENTS>
      </AR-PACKAGE>
   </AR-PACKAGES>
</AR-PACKAGE>
```

コード 21: ARXML コード – アプリケーションデータ型 `UInt8_16` と実装データ型のマッピング（AUTOSAR R4.0.2）

参照される実装データ型はプラットフォーム型ではないため、`Swc_impltypes.arxml` ファイル内に含まれています。

```
<AR-PACKAGE>
   <SHORT-NAME>ImplementationDataTypes</SHORT-NAME>
   <ELEMENTS>
      <IMPLEMENTATION-DATA-TYPE>
         <SHORT-NAME>uint8_16</SHORT-NAME>
         <!--
            array of 16 &quot;uint8&quot; values
         -->
         <CATEGORY>ARRAY</CATEGORY>
         <SUB-ELEMENTS>
            <IMPLEMENTATION-DATA-TYPE-ELEMENT>
               <SHORT-NAME>elementName</SHORT-NAME>
               <CATEGORY>TYPE_REFERENCE</CATEGORY>
               <ARRAY-SIZE>16</ARRAY-SIZE>
               <ARRAY-SIZE-SEMANTICS>FIXED-SIZE</ARRAY-SIZE-SEMANTICS>
               <SW-DATA-DEF-PROPS>
                  <SW-DATA-DEF-PROPS-VARIANTS>
                     <SW-DATA-DEF-PROPS-CONDITIONAL>
                        <IMPLEMENTATION-DATA-TYPE-REF DEST="IMPLEMENTATION-DATA-TYPE">
                        /AUTOSAR_PlatformTypes/ImplementationDataTypes/uint8
                        </IMPLEMENTATION-DATA-TYPE-REF>
                     </SW-DATA-DEF-PROPS-CONDITIONAL>
                  </SW-DATA-DEF-PROPS-VARIANTS>
               </SW-DATA-DEF-PROPS>
            </IMPLEMENTATION-DATA-TYPE-ELEMENT>
         </SUB-ELEMENTS>
      ...
   </ELEMENTS>
</AR-PACKAGE>
```

コード 22: ARXML コード – 実装データ型 `Record_Impl`（AUTOSAR R4.0.2）

実装データ型はプラットフォーム型 uint8 を参照しています。参照される実装データ型は、AUTOSAR_MOD_PlatformTypes.arxml ファイル内に以下のように示されます。

```
<IMPLEMENTATION-DATA-TYPE>
  <SHORT-NAME>uint8</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">unsigned integer 8bit</L-4>
  </LONG-NAME>
  <CATEGORY>VALUE</CATEGORY>
  <INTRODUCTION>
    <TRACE>
      <SHORT-NAME>PLATFORM013</SHORT-NAME>
      <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
      <P>
        <L-1 L="EN">This standard AUTOSAR type shall be of 8 bit unsigned.</L-1>
      </P>
    </TRACE>
  </INTRODUCTION>
  <SW-DATA-DEF-PROPS>
    <SW-DATA-DEF-PROPS-VARIANTS>
      <SW-DATA-DEF-PROPS-CONDITIONAL>
        <BASE-TYPE-REF DEST="SW-BASE-TYPE">
        /AUTOSAR_PlatformTypes/SwBaseTypes/uint8</BASE-TYPE-REF>
      </SW-DATA-DEF-PROPS-CONDITIONAL>
    </SW-DATA-DEF-PROPS-VARIANTS>
  </SW-DATA-DEF-PROPS>
</IMPLEMENTATION-DATA-TYPE>
```

コード 23: ARXML コード – プラットフォームデータ型 uint8（AUTOSAR R4.0.2）

プラットフォーム型 uint8 は基底型 uint8 を参照しています。後者は AUTOSAR_MOD_PlatformBaseTypes_TC1796.arxml ファイル内に以下のように定義されます。

```
<SW-BASE-TYPE>
  <SHORT-NAME>uint8</SHORT-NAME>
  <LONG-NAME>
    <L-4 L="EN">unsigned integer 8bit</L-4>
  </LONG-NAME>
  <CATEGORY>FIXED_LENGTH</CATEGORY>
  <INTRODUCTION>
    <TRACE>
      <SHORT-NAME>PLATFORM013</SHORT-NAME>
      <CATEGORY>SPECIFICATION_ITEM</CATEGORY>
      <P>
        <L-1 L="EN">This standard AUTOSAR type shall be of 8 bit unsigned.</L-1>
      </P>
    </TRACE>
  </INTRODUCTION>
  <BASE-TYPE-SIZE>8</BASE-TYPE-SIZE>
  <BASE-TYPE-ENCODING>NONE</BASE-TYPE-ENCODING>
</SW-BASE-TYPE>
```

コード 24: ARXML コード – 基底型 uint8（AUTOSAR R4.0.2）

6 インターフェース

1 つのアプリケーションが複数のソフトウェアコンポーネントで構成される場合、各ソフトウェアコンポーネントがデータを交換したり所定のファンクションをトリガしたりするために「通信」を行う必要があります。AUTOSAR ソフトウェアコンポーネント間の通信には「ポート」と「インターフェース」を使用します。インターフェースには以下のタイプがあります。

1. センダ／レシーバ（シグナル渡し） – 6.1 項参照
2. モードスイッチ（モードスイッチの通信） – 6.2 項参照
3. クライアント／サーバー（関数呼び出し） – 6.3 項参照
4. 適合 – 6.4 項参照
5. NVデータ [1]（不揮発性シグナルの受け渡し） – 6.5 項参照

これらの通信モデルが AUTOSAR の「インターフェース」と呼ばれます。

ソフトウェアコンポーネントのポートにはPポート（提供ポート）とRポート（要求ポート）があり、これらはインターフェースの型により分類されます。インターフェース型は<SENDER-RECEIVER-INTERFACE>、<MODE-SWITCH-INTERFACE>[1] 、<CLIENT-SERVER-INTERFACE>、<CALPRM-INTERFACE>[2] / <PARAMETER-INTERFACE>[1] 、<NV-DATA-INTERFACE>[1] のいずれかのエレメントを用いて定義されます。

各インターフェース（センダ／レシーバ、クライアント／サーバー、適合）の定義については、本章で詳しく説明します。

ソフトウェアコンポーネントとインターフェースとのインタラクションの方法は、ソフトウェアコンポーネントを参照する<INTERNAL-BEHAVIOR>エレメントにより定義されます。これについては、第 8 章「内部ビヘイビア（Internal Behavior）」で説明します。

6.1 センダ／レシーバ

「センダ／レシーバ通信」においては、アトミックデータエレメントからなるシグナルを 1 つのコンポーネントが送信して 1 つまたは複数のコンポーネントが受信します。

各センダ／レシーバインターフェースには複数のデータエレメントを含めることができ、各データエレメントを個別に送受信できます。

センダ／レシーバインターフェースを作成する:

- コンポーネントマネージャで、**Insert → AUTOSAR → SenderReceiver_Interface** を選択します。
- このセンダ／レシーバインターフェースの名前を SRInterface にします。

ASCET は、AUTOSAR プロジェクト用のコードを生成する際、ファイル Swc_interfaces.arxml 内に<SENDER-RECEIVER-INTERFACE>エレメントを定義します。<SENDER-RECEIVER-INTERFACE>エレメントはコンフィギュレーション言語を用いて以下のような構造で記述されます。

[1] AUTOSAR 4.0.*

[2] AUTOSAR R3.1.5 以前

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_interfaces</SHORT-NAME>
  <DESC></DESC>
  <SUB-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Impl</SHORT-NAME>
      <DESC></DESC>
      <ELEMENTS>
        ...
        <SENDER-RECEIVER-INTERFACE>
          <SHORT-NAME>SRInterface</SHORT-NAME>
          <IS-SERVICE>false</IS-SERVICE>
          <DATA-ELEMENTS>
            ...
          </DATA-ELEMENTS>
          <MODE-GROUPS>
            ...
          </MODE-GROUPS>
        </SENDER-RECEIVER-INTERFACE>
      </ELEMENTS>
    </AR-PACKAGE>
  </SUB-PACKAGES>
</AR-PACKAGE>
```

コード 25: ARXML コード – センダ／レシーバインターフェースの定義（AUTOSAR R3.1.2）

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Interfaces</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Impl</SHORT-NAME>
      <ELEMENTS>
        ...
        <SENDER-RECEIVER-INTERFACE>
          <SHORT-NAME>SRInterface</SHORT-NAME>
          <IS-SERVICE>false</IS-SERVICE>
          <DATA-ELEMENTS>
            ...
          </DATA-ELEMENTS>
        </SENDER-RECEIVER-INTERFACE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 26: ARXML コード – センダ／レシーバインターフェースの定義（AUTOSAR R4.0.*）

センダ／レシーバインターフェースの名前は<SHORT-NAME>で定義されます。この名前は、このインターフェース型を参照するエレメント内で使用されます。

センダ／レシーバインターフェースのショートネームは有効な C 識別子である必要があります。

センダ／レシーバインターフェースはデータ（<DATA-ELEMENTS>エレメント内のデータエレメントプロトタイプを使用）とモード（6.2 項の「 モードスイッチ」を参照）の両方の通信に使用できます。

注記

AUTOSAR **R3.1.5** 以前のバージョンでは、センダ／レシーバインターフェースには、0 個以上のデータエレメント、または 0 個以上のモードグループを定義できますが、データ転送用のインターフェースとモード管理用のインターフェースは分けておくことをお奨めします。

AUTOSAR R4.0.* では、センダ／レシーバインターフェースには、データエレメント、または 1 つのモードグループの、**いずれか一方のみ**を含める必要があります。両方が含まれているとコード生成時にエラーが発生します。

6.1.1 データエレメントプロトタイプ

各センダ／レシーバインターフェースには、そのインターフェース経由で通信される AUTOSAR シグナルからなるデータエレメントを 0 個以上定義できます。各データアイテムは特定の型（基本データ型、RECORD 型、ARRAY 型）のプロトタイプを定義します。データ型の定義についての詳細は、第 4 章「データ型」を参照してください。

ASCET でデータエレメントを作成する:

- コンポーネントマネージャで、`SRInterface` をダブルクリックします。
  “Sender Receiver Interface Editor for: SRInterface”エディタが開きます。
- **Signed Discrete Variable** ボタンを使用して `sdisc` 変数を作成します。
  “Properties for Scalar Element: sdisc”ダイアログボックスが開きます。
- この符号付き離散変数の名前を `Speed` にします。

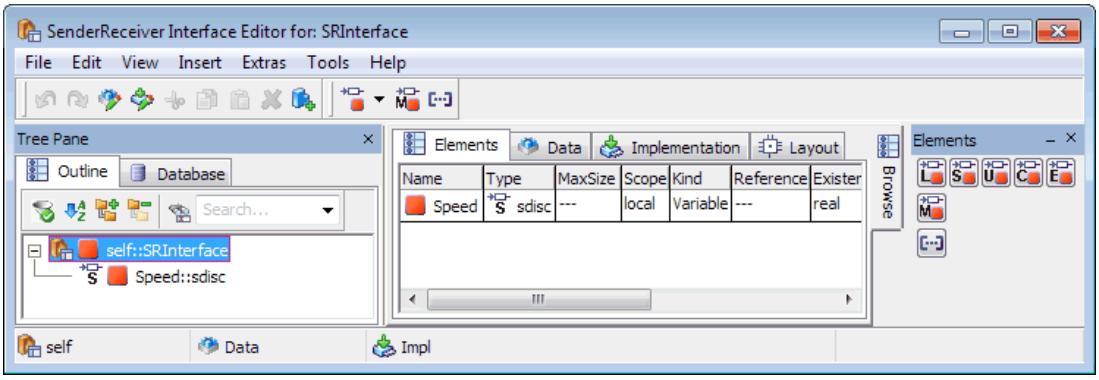


図 17: センダ／レシーバインターフェース“SRInterface”用のデータエレメント“Speed”

データエレメントのインプリメンテーションを作成する:

- “Sender Receiver Interface Editor for: SRInterface”エディタの“Implementation”タブを選択します。
- “Implementation”タブで、`Speed` エレメントをダブルクリックします。
  “Implementation for: Speed”ダイアログボックスが開きます。
- “Master”フィールドで **Implementation** をオンにします。
- “Implementation”フィールドで `sint16` を選択します。
- “Min”／“Max”フィールドを右クリックして、ショートカットメニューから **Default Value** を選択します。
- **OK** をクリックして“Implementation for: sdisc”ダイアログボックスを閉じます。
  “Sender Receiver Interface Editor for: SRInterface”ウィンドウの“Implementation”タブの内容は下図のようになります。

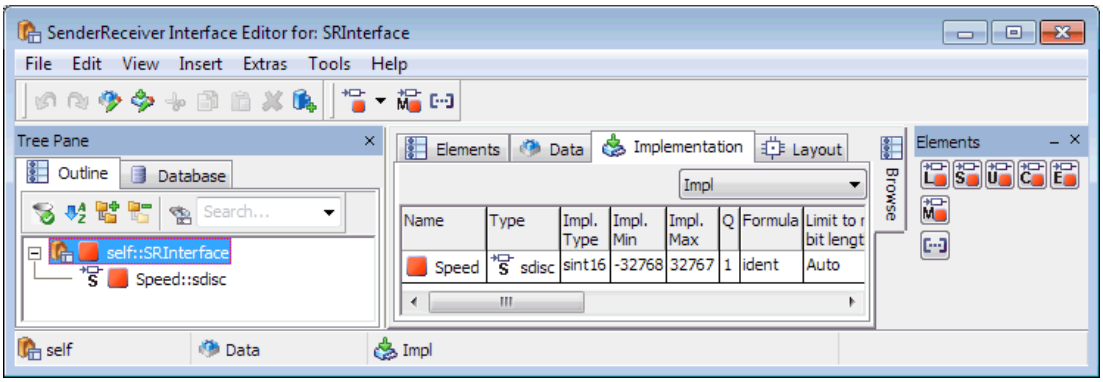


図 18: データエレメント Speed を持つセンダ／レシーバインターフェース SRInterface のインプリメンテーション Impl

ASCET のセンダ／レシーバインターフェースのインプリメンテーションは、AUTOSAR のセンダ／レシーバインターフェースに相当します。コンフィギュレーション言語のセンダ／レシーバインターフェースは ASCET によりファイル Swc_interfaces.arxml 内に生成されます。

AUTOSAR R3.1.5 以前のバージョンでは、センダ／レシーバインターフェース定義内のデータエレメントの宣言は以下のような構造です。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_interfaces</SHORT-NAME>
  <DESC></DESC>
  <SUB-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Impl</SHORT-NAME>
      <DESC></DESC>
      <ELEMENTS>
        <SENDER-RECEIVER-INTERFACE>
          <SHORT-NAME>SRInterface</SHORT-NAME>
          <IS-SERVICE>false</IS-SERVICE>
          <DATA-ELEMENTS>
            <DATA-ELEMENT-PROTOTYPE>
              <SHORT-NAME>Speed</SHORT-NAME>
              <SW-DATA-DEF-PROPS>
                <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
              </SW-DATA-DEF-PROPS>
              <TYPE-TREF DEST="INTEGER-TYPE">/AUTOSAR_types/SInt16
              </TYPE-TREF>
              <IS-QUEUED>false</IS-QUEUED>
            </DATA-ELEMENT-PROTOTYPE>
          </DATA-ELEMENTS>
        </SENDER-RECEIVER-INTERFACE>
      </ELEMENTS>
    </AR-PACKAGE>
  </SUB-PACKAGES>
</AR-PACKAGE>
```

コード 27: ARXML コード ― センダ／レシーバインターフェース内のデータエレメント宣言（AUTOSAR R3.1.2）

AUTOSAR R4.0.*では、センダ／レシーバインターフェース定義内のデータエレメントの宣言は以下のような構造です。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Interfaces</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Impl</SHORT-NAME>
      <ELEMENTS>
        ...
        <SENDER-RECEIVER-INTERFACE>
          <SHORT-NAME>SRInterface</SHORT-NAME>
          <IS-SERVICE>false</IS-SERVICE>
          <DATA-ELEMENTS>
            <VARIABLE-DATA-PROTOTYPE>
              <SHORT-NAME>Speed</SHORT-NAME>
              <SW-DATA-DEF-PROPS>
                <SW-DATA-DEF-PROPS-VARIANTS>
                  <SW-DATA-DEF-PROPS-CONDITIONAL>
                    <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
                    /ASCET_AddrMethods/RAM</SW-ADDR-METHOD-REF>
                    <SW-CALIBRATION-ACCESS>READ-ONLY
                    </SW-CALIBRATION-ACCESS>
                    <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
                  </SW-DATA-DEF-PROPS-CONDITIONAL>
                </SW-DATA-DEF-PROPS-VARIANTS>
              </SW-DATA-DEF-PROPS>
              <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
              /ASCET_Types/ApplicationDataTypes/SInt16</TYPE-TREF>
            </VARIABLE-DATA-PROTOTYPE>
          </DATA-ELEMENTS>
        </SENDER-RECEIVER-INTERFACE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 28: ARXML コード― センダ／レシーバインターフェース内のデータエレメント宣言（AUTOSAR R4.0.2）

データエレメントは<DATA-ELEMENT-PROTOTYPE>[1] / <VARIABLE-DATA-PROTOTYPE>[2]エレメントを使用して定義されます。すべてのエレメントを<DATA-ELEMENTS>エレメント内に定義してカプセル化する必要があります。

各<DATA-ELEMENT-PROTOTYPE>/<VARIABLE-DATA-PROTOTYPE>エレメントには以下のものが定義されている必要があります。

- 当該アイテムを参照する際に使用する<SHORT-NAME>
- データプロパティ<SW-DATA-DEF-PROPS>内の以下のエントリ
- <SW-CALIBRATION-ACESS>
- データアイテムの型への参照 <TYPE-TREF>
- AUTOSAR R3.1.5 以前のみ: 受信データのキューイングの有無を示す<IS-QUEUED>
  - <IS-QUEUED>false</IS-QUEUED> ― 新たに受信した値を以前の値に上書きします。1 つの値が受信される前に新しい値を複数回送信されてしまうと、レシーバは最後に送信された値にしかアクセスできません。

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

– <IS-QUEUED>true</IS-QUEUED> ― センダ／レシーバインターフェースはレシーバ側へのデータ到着をキューイングします。

注記

ASCET V6.2 ではキューイング通信をモデリングすることはできませんが、ASCET の C コードコンポーネントでキューイング通信を実現することは可能です。

6.2 モードスイッチ

AUTOSAR システムは、1 つまたは複数のアプリケーションモードで動作するように設定することができます。1 つのモードスイッチインターフェースには、アプリケーションモードを定義するモードグループを 0 個以上定義できます。

ASCET においてモードスイッチインターフェースは、モードグループを含むセンサ／レシーバインターフェースコンポーネントとして実現されます。

AUTOSAR R3.1.5 以前では、1 つのセンダ／レシーバインターフェースコンポーネントにデータエレメントと モードグループの両方を同じように宣言することができますが、実際には、1 つのインターフェースにはデータエレメントまたはモードグループのいずれかのみを使用することをお勧めします。

AUTOSAR R4.0 以降では、1 つのセンダ／レシーバインターフェースにはデータエレメントとモードグループのいずれか一方しか含めることはできません。両方のエレメントを混在させると、コード生成エラーが発生します。

モードグループを作成する:

- コンポーネントマネージャで、**Insert → AUTOSAR →Mode Group** を選択します。
- このモードグループの名前を `OnOffMode` にします。
- “1 Database”または“1 Workspace”ペインで `OnOffMode` を選択し、“Contents”ペインを選択します。
- **Mode → Rename** を選択してモード名を `off` に変更します。
- **Mode → Add Mode → As Last** を選択して新しいモード on を追加します。

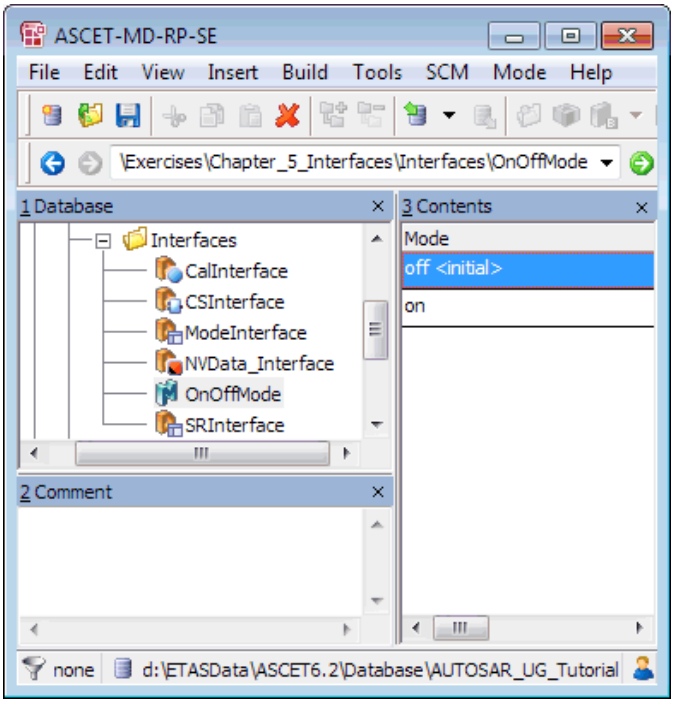


図 19: モード宣言グループ `OnOffMode`

AUTOSAR R3.1.5 以前の場合、ASCET は<MODE-DECLARATION-GROUP>を autosar_types.arxml ファイルの AUTOSAR パッケージ ASCET_types 内に宣言します。

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_types</SHORT-NAME>
   <DESC></DESC>
   <ELEMENTS>
      <MODE-DECLARATION-GROUP>
         <SHORT-NAME>OnOffMode</SHORT-NAME>
         <INITIAL-MODE-REF DEST="MODE-DECLARATION">/ASCET_types/OnOffMode/off
         </INITIAL-MODE-REF>
         <MODE-DECLARATIONS>
            <MODE-DECLARATION>
               <SHORT-NAME>off</SHORT-NAME>
            </MODE-DECLARATION>
            <MODE-DECLARATION>
               <SHORT-NAME>on</SHORT-NAME>
            </MODE-DECLARATION>
         </MODE-DECLARATIONS>
      </MODE-DECLARATION-GROUP>
   </ELEMENTS>
   ...
</AR-PACKAGE>
```

コード 29: ARXML コード ― モード宣言グループ（AUTOSAR R3.1.2）

AUTOSAR R4.0.*の場合、ASCET は<*swc name*>_appltypes.arxml ファイルの AUTOSAR パッケージ ASCET_types 下のサブパッケージ ApplicationDataTypes 内に<MODE-DECLARATION-GROUP>を宣言します。

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_Types</SHORT-NAME>
   <AR-PACKAGES>
      <AR-PACKAGE>
         <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
         <ELEMENTS>
            <MODE-DECLARATION-GROUP>
               <SHORT-NAME>OnOffMode</SHORT-NAME>
               <INITIAL-MODE-REF DEST="MODE-DECLARATION">
               /ASCET_Types/ApplicationDataTypes/OnOffMode/off
               </INITIAL-MODE-REF>
               <MODE-DECLARATIONS>
                  <MODE-DECLARATION>
                     <SHORT-NAME>off</SHORT-NAME>
                  </MODE-DECLARATION>
                  <MODE-DECLARATION>
                     <SHORT-NAME>on</SHORT-NAME>
                  </MODE-DECLARATION>
               </MODE-DECLARATIONS>
            </MODE-DECLARATION-GROUP>
            ...
         </ELEMENTS>
      </AR-PACKAGE>
   </AR-PACKAGES>
</AR-PACKAGE>
```

コード 30: ARXML コード ― モード宣言グループ（AUTOSAR R4.0.2）

モードスイッチインターフェースを作成する:

注記

AUTOSAR R3.1.5 以前では、1 つのインターフェースに複数のモードグループを含めることができますが、AUTOSAR R4.0.*では、1 つのモードしか含めることができません。

- コンポーネントマネージャで、**Insert → AUTOSAR → SenderReceiver Interface** を選択します。
- このセンダ／レシーバインターフェースの名前を `ModeInterface` にします。
- `ModeInterface` をダブルクリックします。
  “Sender Receiver Interface Editor for: ModeInterface”エディタが開きます。
- **Insert → Component** を選択します。
  “Select Item”ダイアログボックスが開きます。

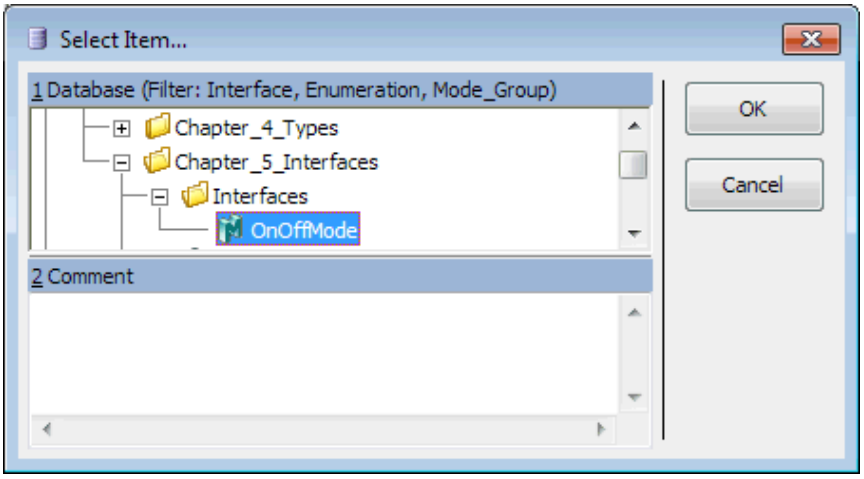


図 20: モードグループ `OnOffMode` を選択する

- “1 Database”または“1 Workspace”ペインで、モードグループ `OnOffMode` を選択します。
- **OK** をクリックして“Select Item”ダイアログボックスを閉じ、`OnOffMode` を `ModeInterface` に挿入します。
  “Properties for Element: OnOffMode”ダイアログボックスが開きます。必要に応じて `OnOffMode` インスタンスの名前とコメントを編集することができます。
- **OK** をクリックしてデフォルトの名前とコメントを確定します。

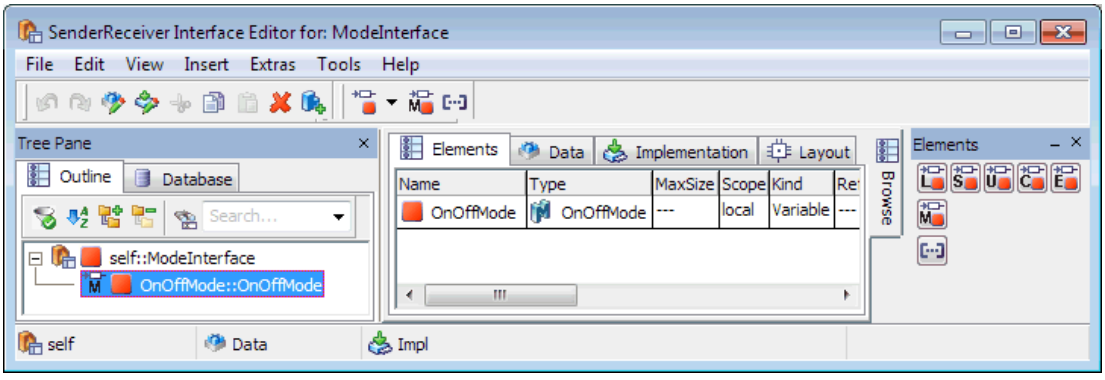


図 21: モードスイッチインターフェース `ModeInterface`

AUTOSAR R3.1.5 以前では、モードスイッチインターフェース（モードグループを含むセンダ／レシーバインターフェース）の定義内のモードグループ宣言は、以下のような構造になります。

```
<SENDER-RECEIVER-INTERFACE>
   <SHORT-NAME>ModeInterface</SHORT-NAME>
   <IS-SERVICE>false</IS-SERVICE>
   <MODE-GROUPS>
      <MODE-DECLARATION-GROUP-PROTOTYPE>
         <SHORT-NAME>OnOffMode</SHORT-NAME>
         <TYPE-TREF DEST="MODE-DECLARATION-GROUP">
         /ASCET_types/OnOffMode</TYPE-TREF>
      </MODE-DECLARATION-GROUP-PROTOTYPE>
   </MODE-GROUPS>
</SENDER-RECEIVER-INTERFACE>
```

コード 31: ARXML コード ― センダ／レシーバインターフェース内のモードグループ宣言（AUTOSAR R3.1.2）

AUTOSAR R4.0.*では、モードスイッチインターフェースの定義内のモードグループ宣言は、以下のような構造になります。

```
<MODE-SWITCH-INTERFACE>
   <SHORT-NAME>R4_ModeInterface</SHORT-NAME>
   <MODE-GROUP>
      <SHORT-NAME>mode_group</SHORT-NAME>
      <TYPE-TREF DEST="MODE-DECLARATION-GROUP">
       /ASCET_Types/ApplicationDataTypes/OnOffMode</TYPE-TREF>
   </MODE-GROUP>
</MODE-SWITCH-INTERFACE>
```

コード 32: ARXML コード― センダ／レシーバインターフェース内のモードグループ宣言–（AUTOSAR R4.0.2）

AUTOSAR R3.1.5 以前では、モードグループ<MODE-DECLARATION-GROUP-PROTOTYPE>エレメントを使用して定義され、全エレメントはカプセル化された<MODE-GROUPS>と全エレメントに、がカプセル化された<MODE-GROUPS>エレメント内に定義されている必要があります。

AUTOSAR R4.0.*では、モードグループは<MODE-GROUP>エレメントを使用して定義します。

各<MODE-DECLARATION-GROUP-PROTOTYPE> / <MODE-GROUP>エレメントで以下のものを定義します。

- 当該アイテムを参照する際に使用する<SHORT-NAME>
- モード宣言グループへの参照<TYPE-TREF>

センダ／レシーバインターフェース内でのモード宣言プロトタイプの使用法については、第 9 章「モード」で詳しく説明します。

6.3 クライアント／サーバー

「クライアント／サーバー通信」では、コンポーネントは別のコンポーネント内の定義済み「サーバー」関数を呼び出します。これらの関数（「オペレーション」と呼ばれます）には、応答が返るものと返らないものとがあります。各クライアント／サーバーインターフェースには複数のオペレーションを定義でき、各オペレーションを別々に呼び出すことができます。

クライアント／サーバーインターフェースを作成する:

- コンポーネントマネージャで、**Insert → AUTOSAR → ClientServer_Interface** を選択します。

- このクライアント／サーバーインターフェースの名前を CSInterface にします。

ASCET は、AUTOSAR プロジェクトにコードを生成する際、ファイル Swc_interfaces.arxml に <CLIENT-SERVER-INTERFACE>エレメントを定義します。<CLIENT-SERVER-INTERFACE>エレメントはコンフィギュレーション言語で以下のような構造で定義されます。

```
<CLIENT-SERVER-INTERFACE>
    <SHORT-NAME>CSInterface</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
    <OPERATIONS>
        ...
    </OPERATIONS>
</CLIENT-SERVER-INTERFACE>
```

コード 33: ARXML コード ― クライアント／サーバーインターフェースの構造（全 AUTOSAR バージョン）

クライアント／サーバーインターフェースの名前は、<SHORT-NAME> エレメントを使用して定義されます。この名前は、このインターフェース型を参照する必要がある他のエレメントの中で使用されます。

クライアント／サーバーインターフェースのショートネームは有効な C 識別子である必要があります。

クライアント／サーバーインターフェースは、<OPERATIONS>コンテナエレメントを使用して定義された 1 つまたは複数のオペレーションで構成されます。

6.3.1 オペレーション

クライアント／サーバーインターフェース内の「オペレーション」は 0 個以上の引数を取ることができます。オペレーションの戻り値は、オペレーションがアプリケーションエラーを返すかどうかに応じて、Std_ReturnType 型か列挙型のいずれかです。

オペレーションを作成する:

- コンポーネントマネージャで、CSInterface をダブルクリックします。
  “Interface Editor for: CSInterface”エディタが開きます。
- “Outline”タブで、ダイアグラム Main を選択します。
- **Insert → Method Signature** を選択します。
  オペレーションが追加されます。
- このオペレーションの名前を MaximumValue にします。

オペレーションの引数を作成する:

- オペレーション MaximumValue をダブルクリックします。
  “Method Signature Editor for: MaximumValue”ダイアログボックスが開きます
- **Argument → Add** を選択して、第 1 引数の名前を InputA にします。以下のように設定します。
  - Argument Type: sdisc
  - Direction: in
- 第 2 の引数 InputB を作成し、同じ型と方向を設定します。
- **Argument → Add** を選択して、第 3 引数の名前を OutputMaximum にし、以下のように設定します。
  - Argument Type: sdisc
  - Direction: out

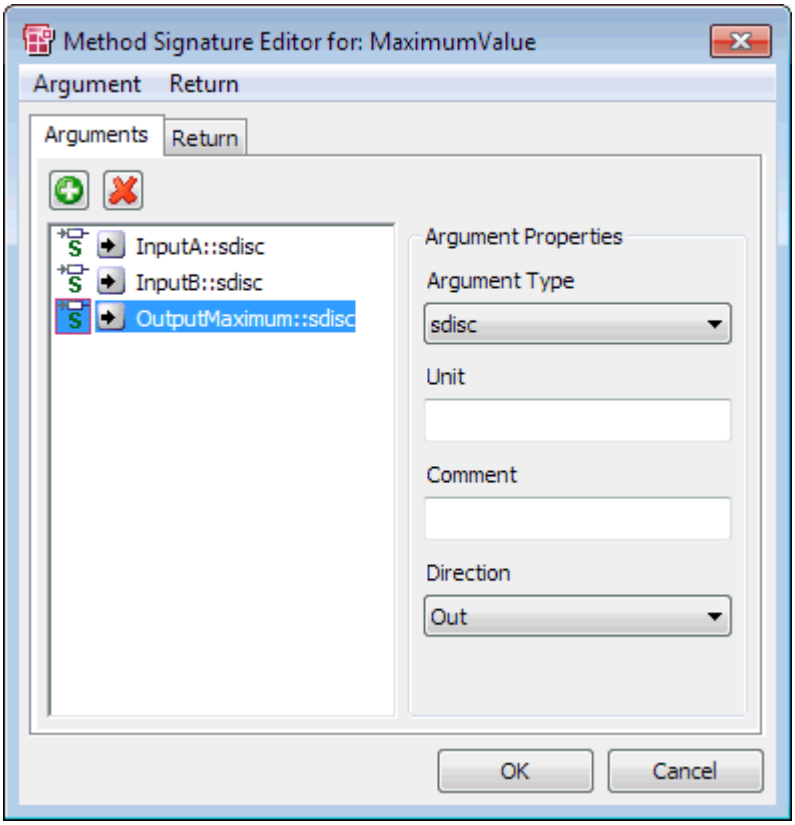


図 22: オペレーション `MaximumValue` の引数

- **OK** をクリックします。
  クライアント／サーバーインターフェース `CSInterface` の内容として、オペレーション `MaximumValue` とその引数（`InputA`、`InputB`、`OutputMaximum`）が下図のように表示されます。

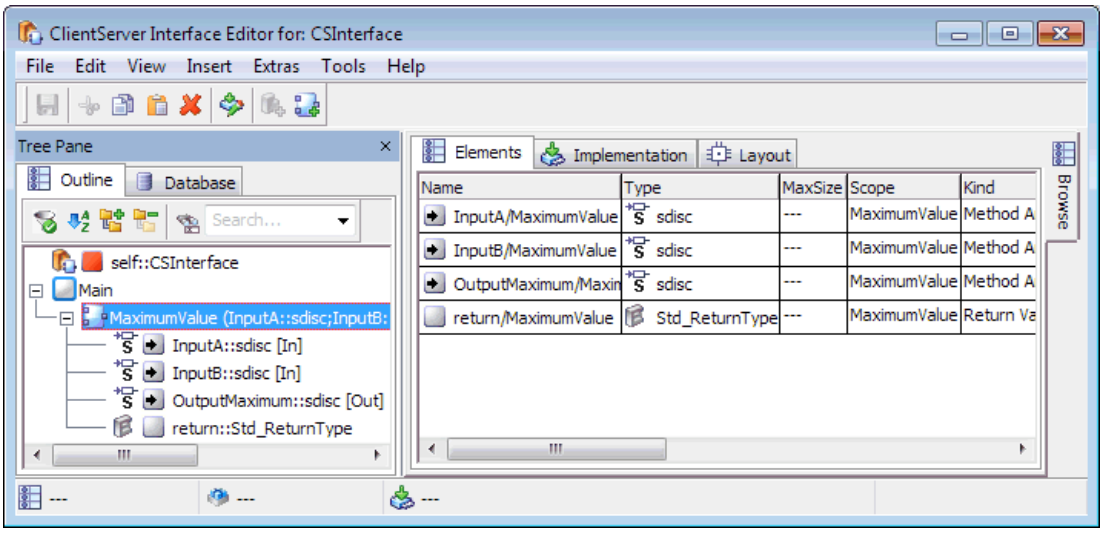


図 23: クライアント／サーバーインターフェース `CSInterface` のオペレーション `MaximumValue`

オペレーションのインプリメンテーションを設定する:

- “Interface Editor for: CSInterface”エディタの“Implementation”タブを選択します。
- “Implementation”タブで、エレメント `InputA` をダブルクリックします。

“Implementation for: InputA”ダイアログボックスが開きます。

- “Master”フィールドで **Implementation** をオンにします。
- “Implementation”フィールドで `sint16` を選択します。
- “Min”／“Max”フィールドを右クリックして、ショートカットメニューから **Default Value** を選択します。
- **OK** をクリックして“Implementation for: InputA”ダイアログボックスを閉じます。
- 引数 `InputB` および `OutputMaximum` について、このインプリメンテーション作成手順を繰り返します。

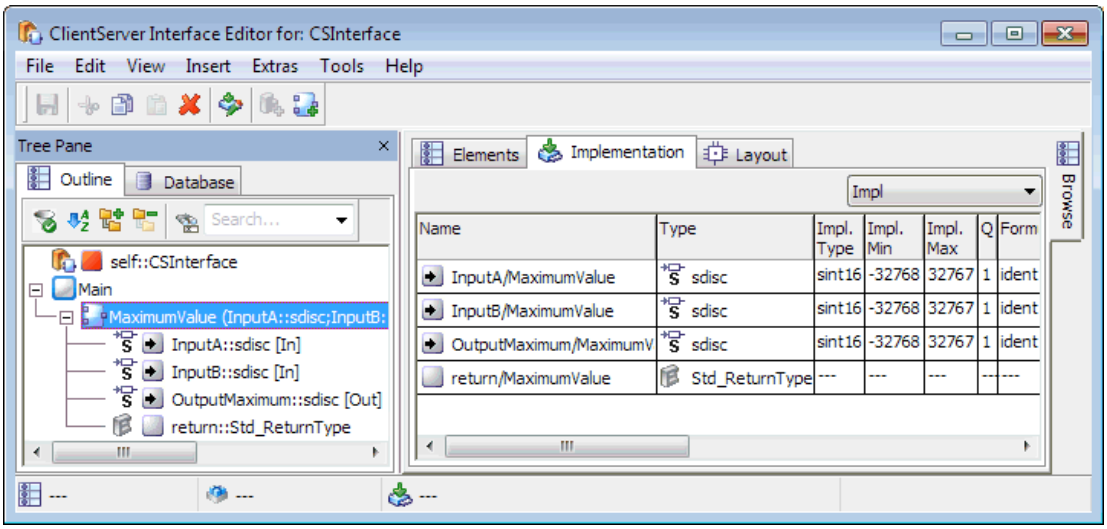


図 24: オペレーション `MaximumValue` のインプリメンテーション

ASCETのクライアント／サーバーインターフェースのインプリメンテーションは、AUTOSARのクライアント／サーバーインターフェースに相当します。コンフィギュレーション言語のクライアント／サーバーインターフェースはASCETによりファイル`Swc_interfaces.arxml`内に生成されます。`<OPERATIONS>`エレメントにより 1 つまたは複数の`<OPERATION-PROTOTYPE>`[1] / `<CLIENT-SERVER-OPERATION>`[2]エレメントがカプセル化され、その各エレメントがクライアント／サーバーインターフェース内の 1 つのオペレーションを定義します。

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

```
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>CSInterface</SHORT-NAME>
  <IS-SERVICE>false</IS-SERVICE>
  <OPERATIONS>
    <OPERATION-PROTOTYPE>
      <SHORT-NAME>MaximumValue</SHORT-NAME>
      <ARGUMENTS>
        <ARGUMENT-PROTOTYPE>
          <SHORT-NAME>InputA</SHORT-NAME>
          <TYPE-TREF DEST="INTEGER-TYPE">/AUTOSAR_types/SInt16</TYPE-TREF>
          <DIRECTION>IN</DIRECTION>
        </ARGUMENT-PROTOTYPE>
        <ARGUMENT-PROTOTYPE>
          <SHORT-NAME>InputB</SHORT-NAME>
          <TYPE-TREF DEST="INTEGER-TYPE">/AUTOSAR_types/SInt16</TYPE-TREF>
          <DIRECTION>IN</DIRECTION>
        </ARGUMENT-PROTOTYPE>
        <ARGUMENT-PROTOTYPE>
          <SHORT-NAME>MaximumValue</SHORT-NAME>
          <TYPE-TREF DEST="INTEGER-TYPE">/AUTOSAR_types/SInt16</TYPE-TREF>
          <DIRECTION>OUT</DIRECTION>
        </ARGUMENT-PROTOTYPE>
      </ARGUMENTS>
    </OPERATION-PROTOTYPE>
  </OPERATIONS>
</CLIENT-SERVER-INTERFACE>
```

コード 34: ARXML コード – クライアント／サーバーインターフェース内のオペレーション（AUTOSAR R3.1.2）

```
<CLIENT-SERVER-INTERFACE>
  <SHORT-NAME>CSInterface</SHORT-NAME>
  <IS-SERVICE>false</IS-SERVICE>
  <OPERATIONS>
    <CLIENT-SERVER-OPERATION>
      <SHORT-NAME>MaximumValue</SHORT-NAME>
      <ARGUMENTS>
        <ARGUMENT-DATA-PROTOTYPE>
          <SHORT-NAME>InputA</SHORT-NAME>
          <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
          /ASCET_Types/ApplicationDataTypes/SInt16</TYPE-TREF>
          <DIRECTION>IN</DIRECTION>
        </ARGUMENT-DATA-PROTOTYPE>
        <ARGUMENT-DATA-PROTOTYPE>
          <SHORT-NAME>InputB</SHORT-NAME>
          <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
          /ASCET_Types/ApplicationDataTypes/SInt16</TYPE-TREF>
          <DIRECTION>IN</DIRECTION>
        </ARGUMENT-DATA-PROTOTYPE>
        <ARGUMENT-DATA-PROTOTYPE>
          <SHORT-NAME>OutputMaximum</SHORT-NAME>
          <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
          /ASCET_Types/ApplicationDataTypes/SInt16</TYPE-TREF>
          <DIRECTION>OUT</DIRECTION>
        </ARGUMENT-DATA-PROTOTYPE>
      </ARGUMENTS>
    </CLIENT-SERVER-OPERATION>
  </OPERATIONS>
</CLIENT-SERVER-INTERFACE>
```

コード 35: ARXML コード – クライアント／サーバーインターフェース内のオペレーション（AUTOSAR R4.0.2）

各オペレーションの名前は<SHORT-NAME>エレメントで定義されます。ここで定義された名前は、RTE がユーザーコード内のオペレーションを参照するために使用する名前の一部になります。

<ARGUMENTS>エレメントにより、オペレーションの各引数を定義する 1 つまたは複数の <ARGUMENT-PROTOTYPE>[1] / <ARGUMENT-DATA-PROTOTYPE>[2]エレメントがカプセル化されます。

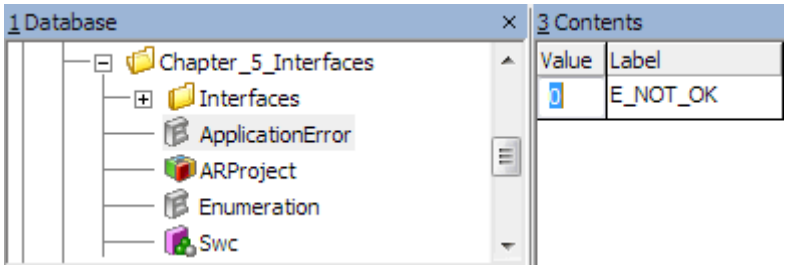
各<ARGUMENT-PROTOTYPE>/<ARGUMENT-DATA-PROTOTYPE>エレメントには以下のものが定義されている必要があります。

- 当該アイテムを参照する際に使用する<SHORT-NAME>
- パラメータの型への参照 <TYPE-TREF>。参照される型は定義済みのものである必要があります。詳しくは第 4 章「データ型」を参照してください。
- パラメータの方向を示す<DIRECTION> ー "IN"（読取りのみ）、"OUT"（書込みのみ）、"INOUT"（コンポーネントによる読み書きが可能）

特に定義されていない場合、クライアント／サーバーインターフェース内のオペレーションは RTE の標準戻り型 Std_ReturnType を返します。また、起こりうるすべてのエラーが定義されている ASCET 列挙型データを選択することにより、アプリケーションエラーを返すことも可能です。

アプリケーションエラーで発生しうるエラーが定義された列挙型データを作成する:

- コンポーネントマネージャで、**Insert → Enumeration** を選択するか、または **Enumeration** ボタンをクリックします。
- この列挙型データの名前を ApplicationError にします。
- この列挙型データを選択します。
  “Contents”ペインに ApplicationError の内容が表示されます。
- “Contents”ペインで列挙子を選択します。
- **Enumeration →Rename** を選択して、ラベルを E_NOT_OK に変更します。
- 値 0 をダブルクリックします。



- 範囲 2～63 の値を設定します。

注記

アプリケーションエラーの値の範囲は[2..63]です。ASCET のアプリケーションエラー用列挙型にこの範囲外の値（2 より小さい値または 63 より大きい値）が含まれていると、コード生成時にエラーが発生します。

アプリケーションエラーをアプリケーションの戻り値に割り当てる:

- コンポーネントマネージャで、クライアント／サーバーインターフェース CSInterface を選択して **Edit → Open Component** を選択するか、または CSInterface をダブルクリックします。

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

“Interface Editor for: CSInterface”エディタが開きます。

- オペレーションをもう 1 つ作成し(71 ページ参照)、`Notification` という名前にします。
- オペレーション `Notification` をダブルクリックします。
  “Method Signature Editor for: Notification“ダイアログボックスが開きます。
- “Return”タブを選択して“Return Type”コンボボックスを開きます。

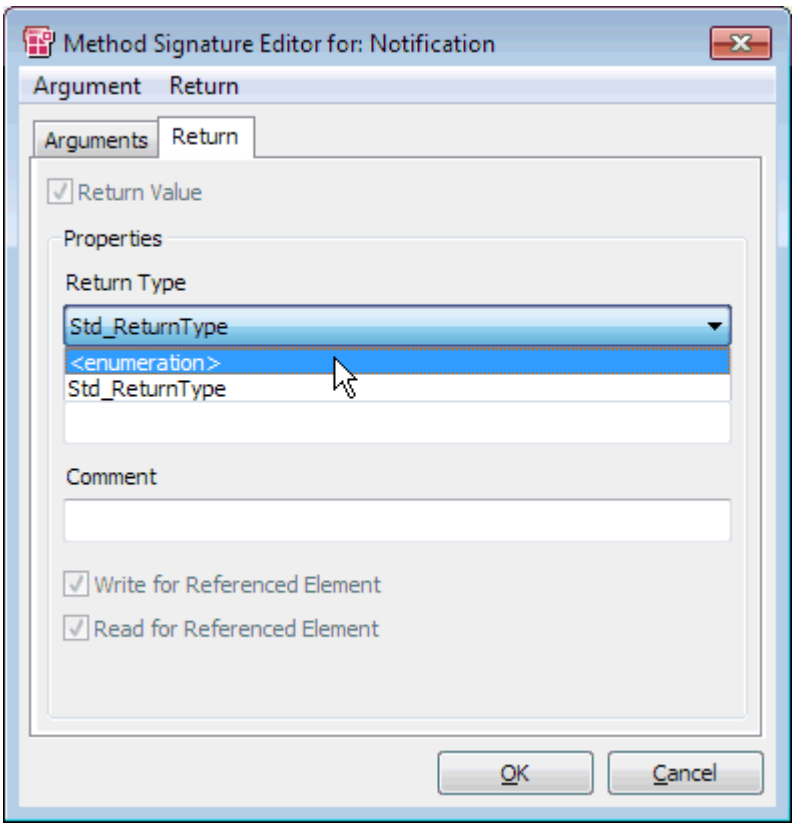
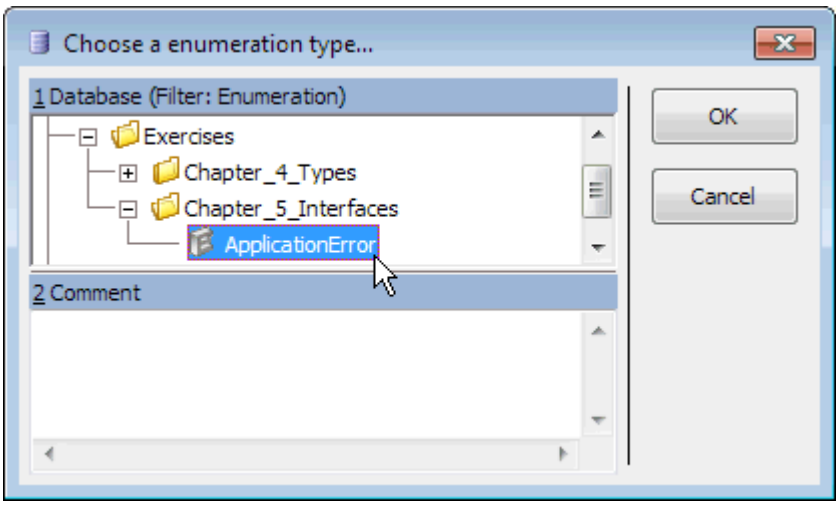


図 25: オペレーション `Notification` の戻り値の型

- `<enumeration>`を選択します。
  “Choose a enumeration type...”ダイアログボックスが開きます。



- 列挙型データ `ApplicationError` を選択します。

- **OK** をクリックして“Choose a enumeration type...”ダイアログボックスを閉じます。
- **OK** をクリックして“Method Signature Editor for: Notification”ダイアログボックスを閉じます。

コンフィギュレーション言語のオペレーション `Notification`、および起こりうるアプリケーションエラーは、ASCET によりファイル `Swc_interfaces.arxml` ファイル内に以下のように記述されます。

```
<CLIENT-SERVER-INTERFACE>
   <SHORT-NAME>CSInterface</SHORT-NAME>
   <IS-SERVICE>false</IS-SERVICE>
   <OPERATIONS>
      <OPERATION-PROTOTYPE>
         <SHORT-NAME>MaximumValue</SHORT-NAME>
         ...
      </OPERATION-PROTOTYPE>
      <OPERATION-PROTOTYPE>
         <SHORT-NAME>Notification</SHORT-NAME>
         <ARGUMENTS></ARGUMENTS>
         <POSSIBLE-ERROR-REFS>
            <POSSIBLE-ERROR-REF DEST="APPLICATION-ERROR">
            /ASCET_interfaces/Impl/CSInterface/E_NOT_OK
            </POSSIBLE-ERROR-REF>
         </POSSIBLE-ERROR-REFS>
      </OPERATION-PROTOTYPE>
   </OPERATIONS>
   <POSSIBLE-ERRORS>
      <APPLICATION-ERROR>
         <SHORT-NAME>E_NOT_OK</SHORT-NAME>
         <ERROR-CODE>2</ERROR-CODE>
      </APPLICATION-ERROR>
   </POSSIBLE-ERRORS>
</CLIENT-SERVER-INTERFACE>
```

コード 36: ARXML コード ― アプリケーションエラーを含むオペレーション（AUTOSAR R3.1.2）

```
<CLIENT-SERVER-INTERFACE>
   <SHORT-NAME>CSInterface</SHORT-NAME>
   <IS-SERVICE>false</IS-SERVICE>
   <OPERATIONS>
      <CLIENT-SERVER-OPERATION>
         <SHORT-NAME>MaximumValue</SHORT-NAME>
         ...
      </CLIENT-SERVER-OPERATION>
      <CLIENT-SERVER-OPERATION>
         <SHORT-NAME>Notification</SHORT-NAME>
         <ARGUMENTS></ARGUMENTS>
         <POSSIBLE-ERROR-REFS>
            <POSSIBLE-ERROR-REF DEST="APPLICATION-ERROR">
            /ASCET_Interfaces/Impl/CSInterface/E_NOT_OK
            </POSSIBLE-ERROR-REF>
         </POSSIBLE-ERROR-REFS>
      </CLIENT-SERVER-OPERATION>
   </OPERATIONS>
   <POSSIBLE-ERRORS>
      <APPLICATION-ERROR>
         <SHORT-NAME>E_NOT_OK</SHORT-NAME>
         <ERROR-CODE>2</ERROR-CODE>
      </APPLICATION-ERROR>
   </POSSIBLE-ERRORS>
</CLIENT-SERVER-INTERFACE>
```

コード 37: ARXML コード ― アプリケーションエラーを含むオペレーション（AUTOSAR R4.0.2）

注記

アプリケーションエラーの値は `Std_ReturnType` の下位 6 ビットにコード化され、値の範囲は [2..63]です。ASCET のアプリケーションエラー用列挙型にこの範囲外の値（2 より小さい値または 63 より大きい値）が含まれていると、コード生成時にエラーが発生します。

6.4 適合

「適合インターフェース」は、適合コンポーネントとの通信に使用されるものです。「適合コンポーネント」は一種のソフトウェアコンポーネントであり、それぞれ適合情報（パラメータと特性値）により一意に構成されています。

各適合インターフェースには複数の適合パラメータを定義できます。AUTOSAR の適合インターフェースを必要とするソフトウェアコンポーネントのポートは、R ポートに対する RTE API を作成することにより、インターフェース内に定義されている任意のパラメータに個別にアクセスできます。適合コンポーネントは適合インターフェースを提供し、適合パラメータを実装します。

適合インターフェースを作成する:

- コンポーネントマネージャで、**Insert → AUTOSAR →Calibration Interface** を選択します。
- この適合インターフェースの名前を `CalInterface` にします。

ASCETは、AUTOSARプロジェクト用のコードを生成する際、`<CALPRM-INTERFACE>`[1] / `<PARAMETER-INTERFACE>`[2]エレメントをファイル`Swc_interfaces.arxml`内に定義します。

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

<CALPRM-INTERFACE>/<PARAMETER-INTERFACE>エレメントはコンフィギュレーション言語で以下のような構造になっています。

```
<CALPRM-INTERFACE>
    <SHORT-NAME>CalInterface</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
    <CALPRM-ELEMENTS>
        <CALPRM-ELEMENT-PROTOTYPE>
            ...
        </CALPRM-ELEMENT-PROTOTYPE>
        ...
    </CALPRM-ELEMENTS>
</CALPRM-INTERFACE>
```

コード 38: ARXML コード ー 適合インターフェースの構造（AUTOSAR R3.1.2）

```
<PARAMETER-INTERFACE>
    <SHORT-NAME>CalInterface</SHORT-NAME>
    <IS-SERVICE>false</IS-SERVICE>
    <PARAMETERS>
        <PARAMETER-DATA-PROTOTYPE>
            ...
        </PARAMETER-DATA-PROTOTYPE>
        ...
    </PARAMETERS>
</PARAMETER-INTERFACE>
```

コード 39: ARXML コード ー適合インターフェースの構造（AUTOSAR R4.0.2）

適合インターフェースの名前は<SHORT-NAME>エレメントで定義されます。この名前は、このインターフェース型を参照する必要のある他のエレメントの中で使用されます。

適合インターフェースのショートネームは有効な C 識別子である必要があります。

適合インターフェースは、<CALPRM-ELEMENTS>[1] / <PARAMETER-DATA-PROTOTYPE>[2]コンテナエレメントを使用して定義された 1 つまたは複数の適合エレメントで構成されます。

6.4.1 適合パラメータ

適合パラメータを作成する:

- コンポーネントマネージャで、CalInterface をダブルクリックします。
  “Calibration Interface Editor for: CalInterface”エディタが開きます。
-  **Logic Parameter** ボタンを使用して論理パラメータを作成します。
  “Properties for Scalar Element: log”ダイアログボックスが開きます。
- このパラメータの名前を CalParam1 にします。
- 論理パラメータをもう 1 つ作成し、CalParam2 という名前にします。
-  **Unsigned Discrete Parameter** ボタンを使用して、符号なし離散型パラメータを作成します。
  “Properties for Scalar Element: udisc”ダイアログボックスが開きます。

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

- このパラメータの名前を CalParam3 にします。

適合パラメータのインプリメンテーションを作成する:

- “Calibration Interface Editor for: CalInterface”エディタの“Implementation”タブを選択します。
- “Implementation”タブで、CalParam3 エレメントをダブルクリックします。
  “Implementation for: CalParam3” ダイアログボックスが開きます。
- “Master”フィールドで **Model** をオンにします。
- “Model”フィールドのテキストボックス“Max”に値 24 を入力します。
- **OK** をクリックします。
  “Calibration Interface Editor for: CalInterface” ウィンドウの“Implementation”タブの内容は下図のようなります。

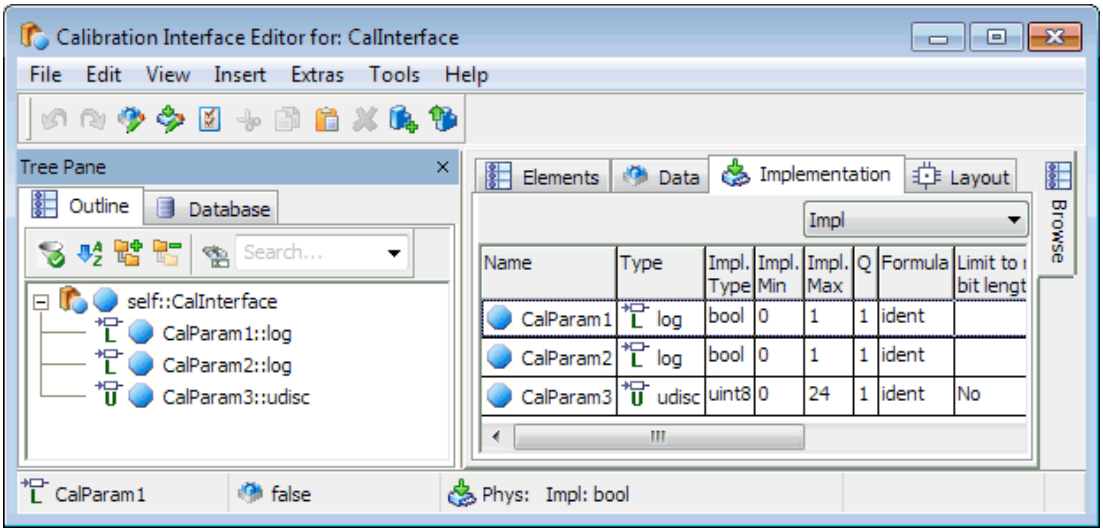


図 26: 適合インターフェース CalInterface のインプリメンテーション Impl

ASCET の適合インターフェースを実装したものは AUTOSAR の適合インターフェースに相当します。適合インターフェースは、ASCET により Swc_interfaces.arxml ファイル内にコンフィギュレーション言語で定義されます。定義された適合インターフェース内には以下のように適合エレメントが宣言されます。

```
<CALPRM-INTERFACE>
  <SHORT-NAME>CalInterface</SHORT-NAME>
  <IS-SERVICE>false</IS-SERVICE>
  <CALPRM-ELEMENTS>
    <CALPRM-ELEMENT-PROTOTYPE>
      <SHORT-NAME>CalParam1</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="BOOLEAN-TYPE">/AUTOSAR_types/Boolean</TYPE-TREF>
    </CALPRM-ELEMENT-PROTOTYPE>
    <CALPRM-ELEMENT-PROTOTYPE>
      <SHORT-NAME>CalParam2</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="BOOLEAN-TYPE">/AUTOSAR_types/Boolean</TYPE-TREF>
    </CALPRM-ELEMENT-PROTOTYPE>
    <CALPRM-ELEMENT-PROTOTYPE>
      <SHORT-NAME>CalParam3</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="INTEGER-TYPE">
      /ASCET_types/artificial/ASCET_Scalar_UInt8_ident_p0p24</TYPE-TREF>
    </CALPRM-ELEMENT-PROTOTYPE>
  </CALPRM-ELEMENTS>
</CALPRM-INTERFACE>
```

コード 40: ARXML コード ― 適合インターフェース定義内の適合エレメント宣言（AUTOSAR R3.1.2）

```
<PARAMETER-INTERFACE>
  <SHORT-NAME>CalInterface</SHORT-NAME>
  <IS-SERVICE>false</IS-SERVICE>
  <PARAMETERS>
    <PARAMETER-DATA-PROTOTYPE>
      <SHORT-NAME>CalParam1</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
            /ASCET_AddrMethods/CAL_MEM</SW-ADDR-METHOD-REF>
            <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
            <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
      /ASCET_Types/ApplicationDataTypes/Boolean</TYPE-TREF>
    </PARAMETER-DATA-PROTOTYPE>
    <PARAMETER-DATA-PROTOTYPE>
      <SHORT-NAME>CalParam2</SHORT-NAME>
      ...
    </PARAMETER-DATA-PROTOTYPE>
    <PARAMETER-DATA-PROTOTYPE>
      <SHORT-NAME>CalParam3</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
            /ASCET_AddrMethods/CAL_MEM</SW-ADDR-METHOD-REF>
            <SW-CALIBRATION-ACCESS>READ-WRITE</SW-CALIBRATION-ACCESS>
            <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
      /ASCET_Types/ApplicationDataTypes/UInt8_ident_p0_p24</TYPE-TREF>
    </PARAMETER-DATA-PROTOTYPE>
  </PARAMETERS>
</PARAMETER-INTERFACE>
```

コード 41: ARXML コード ー適合インターフェース定義内の適合エレメント宣言 definition（AUTOSAR R4.0.2）

適合エレメントは<CALPRM-ELEMENT-PROTOTYPE>[1] / <PARAMETER-DATA-PROTOTYPE>[2]エレメントを使用して定義され、すべての適合エレメントの定義が<CALPRM-ELEMENTS>[1] / <PARAMETERS>[2] エレメント内にカプセル化されます。

各<CALPRM-ELEMENT-PROTOTYPE>/<PARAMETER-DATA-PROTOTYPE>エレメントには以下のものが定義されている必要があります。

- 当該アイテムを参照する際に使用する<SHORT-NAME>
- データプロパティ<SW-DATA-DEF-PROPS>内の以下のエントリ
  - <Calibration-ACESS>
- データアイテムの型への参照 <TYPE-TREF>

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

6.5 NVデータ(AUTOSAR R4.0.*のみ)

AUTOSAR R4.0 で導入された<NV-DATA-INTERFACE>エレメントは、NV ブロックというソフトウェアコンポーネントタイプに使用されるインターフェースを定義するものです。

注記

NV データ(NVData)インターフェースは、AUTOSAR R3.1.5 以前に対応する ASCET プロジェクトでは使用できません。そのようなプロジェクトで使用すると、コード生成時にエラーが発生します。

NV データインターフェースには、複数の NV データエレメントを含めることができ、個々の NV データを送受信することができます。

NV データインターフェースを作成する:

- R4_project という名前のプロジェクトを作成し(23 ページ参照)、コード生成オプションを AUTOSAR R4.0.2 用に設定します(23 ページ参照)。
- SWC という名前のソフトウェアコンポーネントを作成し(26 ページ参照)、R4_project に挿入します(26 ページ参照)。
- R4_project を開き、さらに SWC を開きます。
- コンポーネントマネージャで、**Insert → AUTOSAR →NVData Interface** を選択します。
- NV データインターフェースの名前を NVData_Interface にします。
- NVData_Interface を SWC に挿入します。

AUTOSAR プロジェクトのコードを生成する際、ASCET は Swc_interfaces.arxml ファイル内に 1 つの<NVDATA-INTERFACE>エレメントを生成します。<NVDATA-INTERFACE>エレメントエレメントはコンフィギュレーション言語を用いて以下のような構造で記述されます。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_Interfaces</SHORT-NAME>
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Impl</SHORT-NAME>
      <ELEMENTS>
        <NV-DATA-INTERFACE>
          <SHORT-NAME>NVData_Interface</SHORT-NAME>
          <IS-SERVICE>false</IS-SERVICE>
          <NV-DATAS>
            <VARIABLE-DATA-PROTOTYPE>
              ...
            </VARIABLE-DATA-PROTOTYPE>
            ...
          </NV-DATAS>
        </NV-DATA-INTERFACE>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AR-PACKAGE>
```

コード 42: ARXML コード ― NV データインターフェースの構造(AUTOSAR R4.0.2)

NV データインターフェースの名前は<SHORT-NAME>エレメントで定義されます。この名前は、このインターフェース型を参照する必要のある他のエレメントの中で使用されます。たとえば、上記の

NV データインターフェースを使用するソフトウェアコンポーネントは、NV データインターフェースを NVData_interface という名前で指定します。

NV データインターフェースのショートネームは有効な C 識別子である必要があります。

NV データインターフェースの NV データエレメント(<NV-DATAS>エレメント内の変数データプロトタイプ)を使用することにより、不揮発性データを通信することができます。

6.5.1 変数データプロトタイプ

NV データインターフェースでは 0 個以上の NV データエレメント(変数データプロトタイプ)を定義することができます。これはインターフェースを介して通信される AUTOSAR シグナルを構成するためのものです。各データエレメントは定義された型のプロトタイプを定義し、基本データ型、RECORD 型、ARRAY 型のいずれかを使用できます。データ型についての詳細は第 4 章「データ型」を参照してください。

ASCET の NV データエレメントを設定する:

- “Software Component Editor for: R4_SWC”ウィンドウで、NVData_Interface をダブルクリックします。
  NVData_Interface 用の NV データインターフェースエディタが開きます。
- Speed_NV という名前の sdisc エレメントを作成します(64 ページ参照)。
- Speed_NV のインプリメンテーションを作成し、64 ページで説明されている内容と同じように設定します。

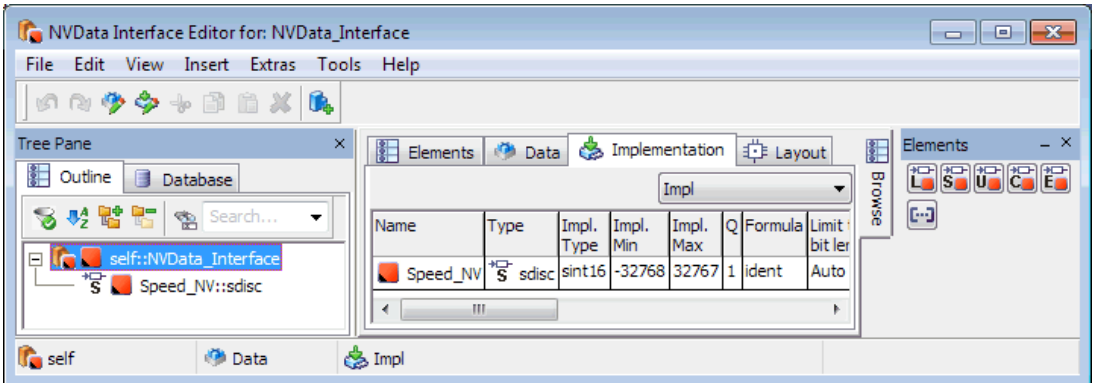


図 27: NV データインターフェース NVData_Interface 内の NV データエレメント Speed_NV (インプリメンテーション:Impl)

ASCET の NV データインターフェースのインプリメンテーションは、AUTOSAR の NV データインターフェースに相当します。コンフィギュレーション言語の NV データインターフェースは ASCET によりファイル Swc_interfaces.arxml 内に生成されます。NV データエレメントは、NV データインターフェース内で以下の構造で定義されます。

```
<NV-DATA-INTERFACE>
  <SHORT-NAME>NVData_Interface</SHORT-NAME>
  <IS-SERVICE>false</IS-SERVICE>
  <NV-DATAS>
    <VARIABLE-DATA-PROTOTYPE>
      <SHORT-NAME>speed_nv</SHORT-NAME>
      <SW-DATA-DEF-PROPS>
        <SW-DATA-DEF-PROPS-VARIANTS>
          <SW-DATA-DEF-PROPS-CONDITIONAL>
            <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">
            /ASCET_AddrMethods/RAM</SW-ADDR-METHOD-REF>
            <SW-CALIBRATION-ACCESS>READ-ONLY
            </SW-CALIBRATION-ACCESS>
            <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
          </SW-DATA-DEF-PROPS-CONDITIONAL>
        </SW-DATA-DEF-PROPS-VARIANTS>
      </SW-DATA-DEF-PROPS>
      <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
      /ASCET_Types/ApplicationDataTypes/SInt16</TYPE-TREF>
    </VARIABLE-DATA-PROTOTYPE>
  </NV-DATAS>
</NV-DATA-INTERFACE>
```

コード 43: ARXML コード ― NV データインターフェース内の NV データエレメント（AUTOSAR R4.0.2）

NV データエレメントは、<NV-DATAS>エレメント内の<VARIABLE-DATA-PROTOTYPE>エレメントで定義されます。

各<VARIABLE-DATA-PROTOTYPE>エレメントには以下のものが定義されている必要があります。

- 当該アイテムを参照する際に使用する<SHORT-NAME>
- データプロパティ<SW-DATA-DEF-PROPS>内の以下のエントリ
  - <SW-ADDR-METHOD-REF>
  - <SW-CALIBRATION-ACESS>
  - <SW-IMPL-POLICY>
- データアイテムの型への参照 <TYPE-TREF>

7 ソフトウェアコンポーネント型

「ソフトウェアコンポーネント」は AUTOSAR のアプリケーションにおけるアトミックソフトウェアユニット（つまり分割不可能な最小単位のソフトウェア）です。ソフトウェアコンポーネントのインタラクションは「ポート」（型が定義されたインターフェース）を通じて行われます。インターフェースはデータの通信可否および通信のセマンティックスを制御します。

AUTOSAR ソフトウェアコンポーネントを作成する:

- コンポーネントマネージャで、**Insert → AUTOSAR →Software Component** を選択します。
- このソフトウェアコンポーネントの名前を Swc にします。
- 3.1.2 項「AUTOSAR 用コード生成に関する設定」で説明されている手順に従って AUTOSAR プロジェクト ARProject を作成し、AUTOSAR コード生成オプションを設定します。
- 26 ページに説明されている方法で、ソフトウェアコンポーネント Swc をこのプロジェクトに挿入します。

AUTOSAR プロジェクト内でソフトウェアコンポーネントを開く:

- コンポーネントマネージャで、プロジェクト ARProject をダブルクリックします。
  プロジェクトエディタウィンドウが開きます。
- プロジェクトエディタウィンドウの“Outline”タブで、ソフトウェアコンポーネント Swc をダブルクリックします。
  ソフトウェアコンポーネントエディタウィンドウが開きます。

各ソフトウェアコンポーネントのコンポーネント型は RTE ジェネレータのコンフィギュレーションで宣言されている必要があります。この「コンポーネント型」により、コンポーネントをさらに大規模なソフトウェアシステムに組み込むことが可能となります。アプリケーションソフトウェアコンポーネント型は *<swc name>*.arxml ファイル内に<APPLICATION-SOFTWARE-COMPONENT-TYPE>エレメントで定義されます。

```
<APPLICATION-SOFTWARE-COMPONENT-TYPE>
  <SHORT-NAME>SWC</SHORT-NAME>
  <PORTS>
    ...
  </PORTS>
</APPLICATION-SOFTWARE-COMPONENT-TYPE>
```

コード 44: ARXML コード – アプリケーションソフトウェアのコンポーネント型（AUTOSAR R3.1.2）

```
<APPLICATION-SW-COMPONENT-TYPE>
  <SHORT-NAME>SWC</SHORT-NAME>
  <PORTS>
    ...
  </PORTS>
  ...
</APPLICATION-SW-COMPONENT-TYPE>
```

コード 45: ARXML コード – アプリケーションソフトウェアのコンポーネント型（AUTOSAR R4.0.2）

ソフトウェアコンポーネント型の名前は<SHORT-NAME>エレメントで定義されます。この名前はシステム内で一意でなければならず、このソフトウェアコンポーネント型を参照する他のエレメント内で使用されます。

ソフトウェアコンポーネントのショートネームは有効な C 識別子である必要があります。

7.1 ポート

「ポート」はソフトウェアコンポーネントからインターフェースへのアクセスを提供するものです。ポートには P ポート（提供ポート）と R ポート（要求ポート）の 2 種類があります。

ソフトウェアコンポーネントのポートは<PORTS>エレメントで定義されます。

```
<PORTS>
    <R-PORT-PROTOTYPE>
        ...
    </R-PORT-PROTOTYPE>
    ...
    <P-PORT-PROTOTYPE>
        ...
    </P-PORT-PROTOTYPE>
    ...
</PORTS>
```

コード 46: ARXML コード – ポート定義の構造（全 AUTOSAR バージョン）

<PORTS>エレメント内の<P-PORT-PROTOTYPE>エレメントで P ポートを定義し、<R-PORT-PROTOTYPE>エレメントで R ポートを定義します。2 つのコンポーネントが通信しあう場合、一般に P ポートと R ポートの両方が同じインターフェース定義を参照します。これにより、両コンポーネントの相互の適合性が保証されます。

7.1.1 Pポート（提供ポート）

「P ポート」（Pport：提供ポート）は、ソフトウェアコンポーネントが他のソフトウェアコンポーネントにデータまたはサービスを提供するために使用するものです。P ポートには「センダポート」と「サーバーポート」とがあります。

センダポートを作成する:

- “Software Component Editor for: Swc”ウィンドウで **Insert → Component** を選択します。
  “Select item...”ダイアログボックスが開きます。
- “Select item...”ダイアログボックスの“1 Database”または“1 Workspace”フィールドで、インターフェース `SRInterface` を選択して **OK** をクリックします。

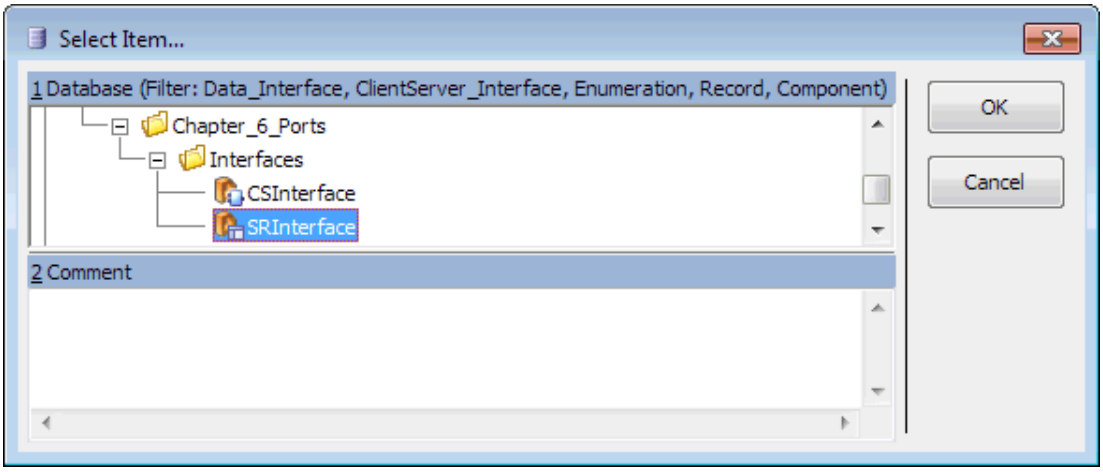


図 28: アイテム `SRInterface` を選択する

“Properties for complex element: SRInterface”ダイアログボックスが開きます。

- このポートの名前を Sender にし、“Internal Access”フィールドで **Provided** をオンにして **OK** をクリックします。

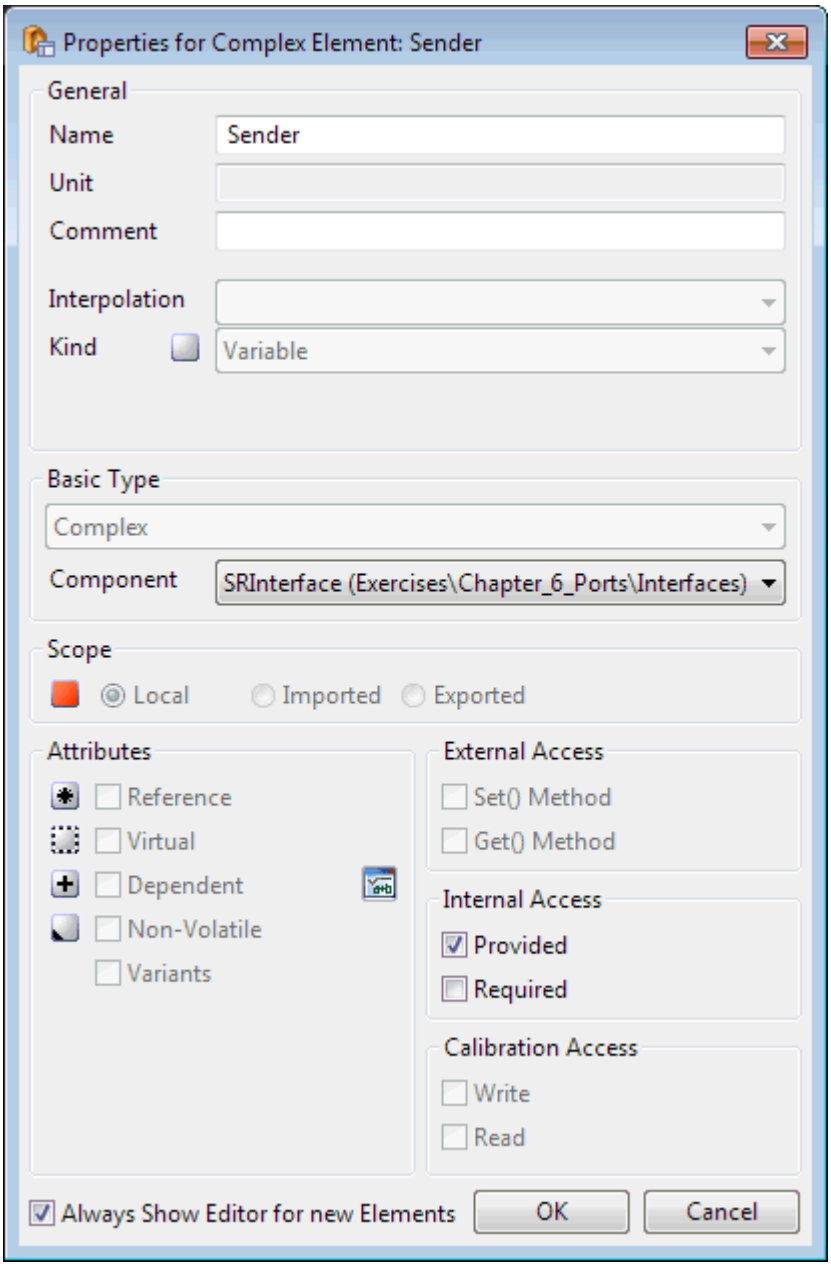


図 29: SRInterface 型の P ポート Sender

- エレメント Sender::SRInterface をソフトウェアコンポーネントエディタの“Outline”タブから描画エリアにドラッグ&ドロップします。

  エレメント Speed を持つ P ポート Sender が、描画エリアに以下のように表示されます。

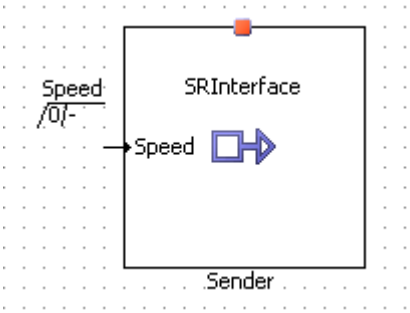


図 30: ソフトウェアコンポーネントエディタの描画エリアに表示された P ポート Sender

ソフトウェアコンポーネントの型定義内で、P ポートの名前は<SHORT-NAME>エレメントで定義されます。この名前は、このソフトウェアコンポーネント型を参照する他のエレメントの中で使用されます。P ポートのショートネームは有効な C 識別子である必要があります。

P ポートの定義においては、他のポートとの通信に使用する定義済みインターフェース型が、*<swc name>*.arxml ファイル内の<PROVIDED-INTERFACE-TREF>エレメントで指定されている必要があります。

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_swcomponents</SHORT-NAME>
  <DESC></DESC>
  <SUB-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>Impl</SHORT-NAME>
      <DESC></DESC>
      <ELEMENTS>
        <APPLICATION-SOFTWARE-COMPONENT-TYPE>
          <SHORT-NAME>Swc</SHORT-NAME>
          <PORTS>
            <P-PORT-PROTOTYPE>
              <SHORT-NAME>Sender</SHORT-NAME>
              <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
              /ASCET_interfaces/Impl/SRInterface
              </PROVIDED-INTERFACE-TREF>
            </P-PORT-PROTOTYPE>
          </PORTS>
        </APPLICATION-SOFTWARE-COMPONENT-TYPE>
        ...
      </ELEMENTS>
    </AR-PACKAGE>
  </SUB-PACKAGES>
</AR-PACKAGE>
```

コード 47: ARXML コード – P ポート Sender の定義（AUTOSAR R3.1.2）

さらに AUTOSAR R4.0.*では、個々のデータエレメントの詳細を含む<PROVIDED-COM-SPECS>エレメントが必要です。以下に詳細情報の一例を示します。

- <DATA-ELEMENT-REF> - データエレメントの識別
- <INIT-VALUE> - データエレメントの初期値

```
<AR-PACKAGE>
  <SHORT-NAME>ASCET_ComponentTypes</SHORT-NAME>
  <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
      <SHORT-NAME>R4_SWC</SHORT-NAME>
      <PORTS>
        <P-PORT-PROTOTYPE>
          <SHORT-NAME>Sender</SHORT-NAME>
          <PROVIDED-COM-SPECS>
            <NONQUEUED-SENDER-COM-SPEC>
              <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
              /ASCET_Interfaces/Impl/SRInterface/Speed</DATA-ELEMENT-REF>
              <INIT-VALUE>
                <NUMERICAL-VALUE-SPECIFICATION>
                  <VALUE>0</VALUE>
                </NUMERICAL-VALUE-SPECIFICATION>
              </INIT-VALUE>
            </NONQUEUED-SENDER-COM-SPEC>
          </PROVIDED-COM-SPECS>
          <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
          /ASCET_Interfaces/Impl/SRInterface</PROVIDED-INTERFACE-TREF>
        </P-PORT-PROTOTYPE>
      </PORTS>
      ...
    </APPLICATION-SW-COMPONENT-TYPE>
  </ELEMENTS>
  ...
</AR-PACKAGE>
```

コード 48: ARXML コード – P ポート Sender の定義（AUTOSAR R4.0.2）

サーバーポートを作成する:

- “Software Component Editor for: Swc”ウィンドウで **Insert → Component** を選択します。
  “Select item...”ダイアログボックスが開きます。
- “Select item...”ダイアログボックスの“1 Database”または“1 Workspace”ペインでインターフェース `CSInterface` を選択して **OK** をクリックします。
  “Properties for complex element: CSInterface”ダイアログボックスが開きます。
- このポートの名前を `Server` にし、“Internal Access”フィールドで **Provided** をオンにして **OK** をクリックします。

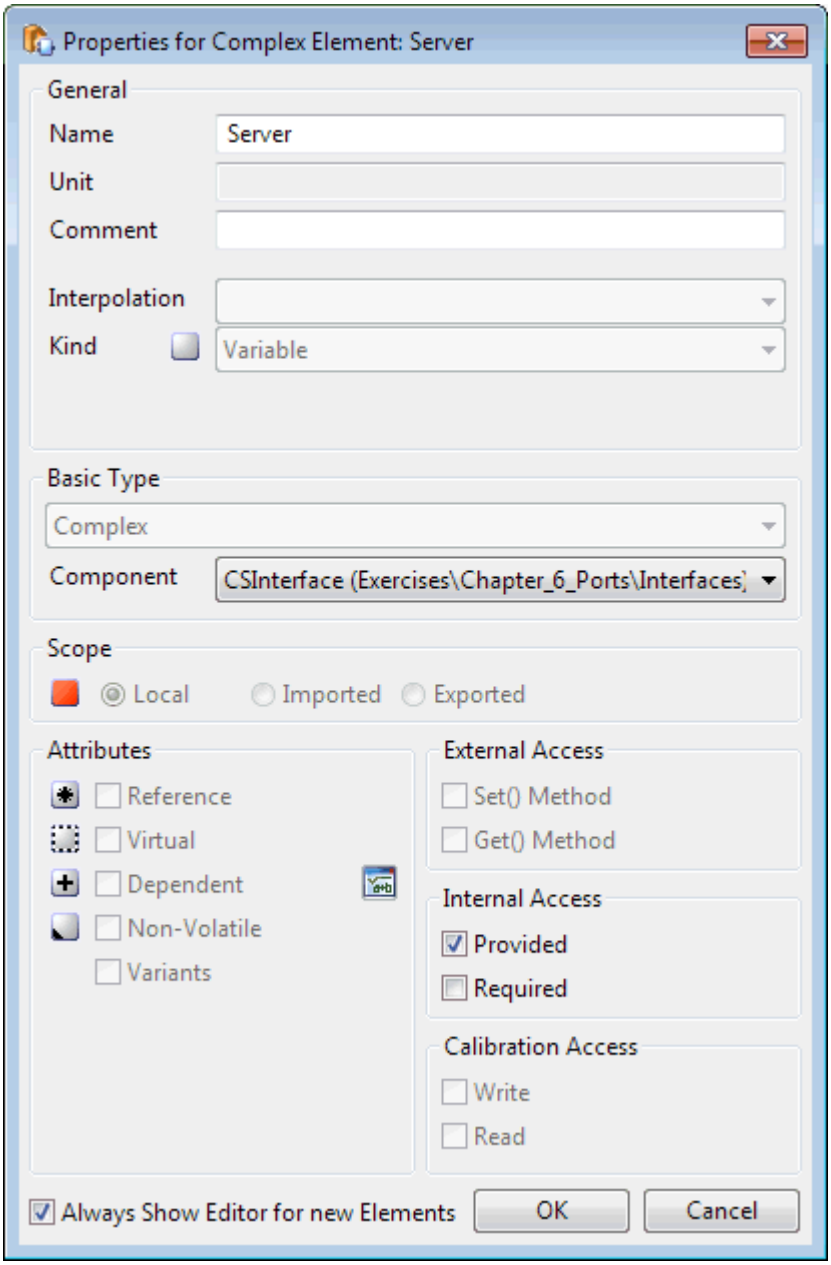


図 31: CSInterface 型の P ポート Server

これにより ASCET は以下のアイテムを作成します。

- Server::CSInterface という名前のサーバーノード（Realized Interfaces フォルダ下）
- ダイアグラム Server_CSInterface
- クライアント／サーバーインターフェース CSInterface 内の各オペレーション用サーバーランナブル（以下の図のように、ランナブル Server_MaximumValue および Server_Notification.が作成されます）

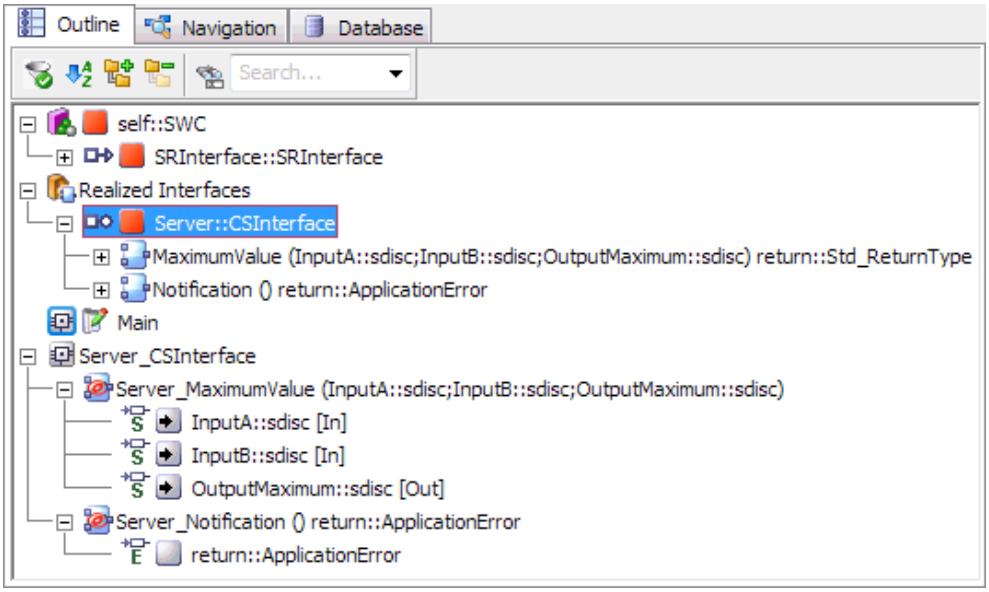


図 32: ソフトウェアコンポーネント Swc の“Outline”タブ内の P ポート Server

サーバーランナブルのエントリ関数の戻り値の型は、そのサーバーがアプリケーションエラーを返す場合は Std_ReturnType、返さない場合は void です。

P ポートの定義では、<PROVIDED-INTERFACE-TREF>エレメント内で、他のポートとの通信に使用する、定義済みインターフェース型への参照を定義する必要があります。

```
<P-PORT-PROTOTYPE>
   <SHORT-NAME>Server</SHORT-NAME>
   <PROVIDED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">
   /ASCET_interfaces/Impl/CSInterface
   </PROVIDED-INTERFACE-TREF>
</P-PORT-PROTOTYPE>
```

コード 49: ARXML コード – P ポート Server の定義（全 AUTOSAR バージョン）

さらに、ASCET はソフトウェアコンポーネント Swc の内部ビヘイビアに情報を追加します。1 つのオペレーションが呼び出すイベントがサーバーポート内の各オペレーション用に作成され、ランナブルエンティティも同様に 1 つずつ作成されます。詳細は第 8 章「内部ビヘイビア（Internal Behavior）」を参照してください。

注記

クライアント／サーバーインターフェースは、サーバーがソフトウェアコンポーネントに挿入される際に変更される可能性があります。この場合、メニューコマンド **Build → Update Interfaces** を実行して、ソフトウェアコンポーネント内のサーバーインターフェースを更新する必要があります。

7.1.2 Rポート（要求ポート）

「R ポート」（RPort：要求ポート）は、ソフトウェアコンポーネントが他のソフトウェアコンポーネントにデータまたはサービスを要求するために使用されるものです。R ポートには「レシーバポート」と「クライアントポート」とがあります。

R ポートの定義は、<R-PORT-PROTOTYPE>エレメントが使用される点以外は P ポートの定義と同じです。

レシーバポートを作成する:

- “Software Component Editor for: Swc”ウィンドウで **Insert → Component** を選択します。
  “Select item...”ダイアログボックスが開きます。
- “Select item...”ダイアログボックスの“1 Database”または“1 Workspace”ペインでインターフェース `SRInterface` を選択して **OK** をクリックします。
  “Properties for complex element: SRInterface”ダイアログボックスが開きます。
- このポートの名前を `Receiver` にし、“Internal Access”フィールドで **Required** をオンにして **OK** をクリックします。

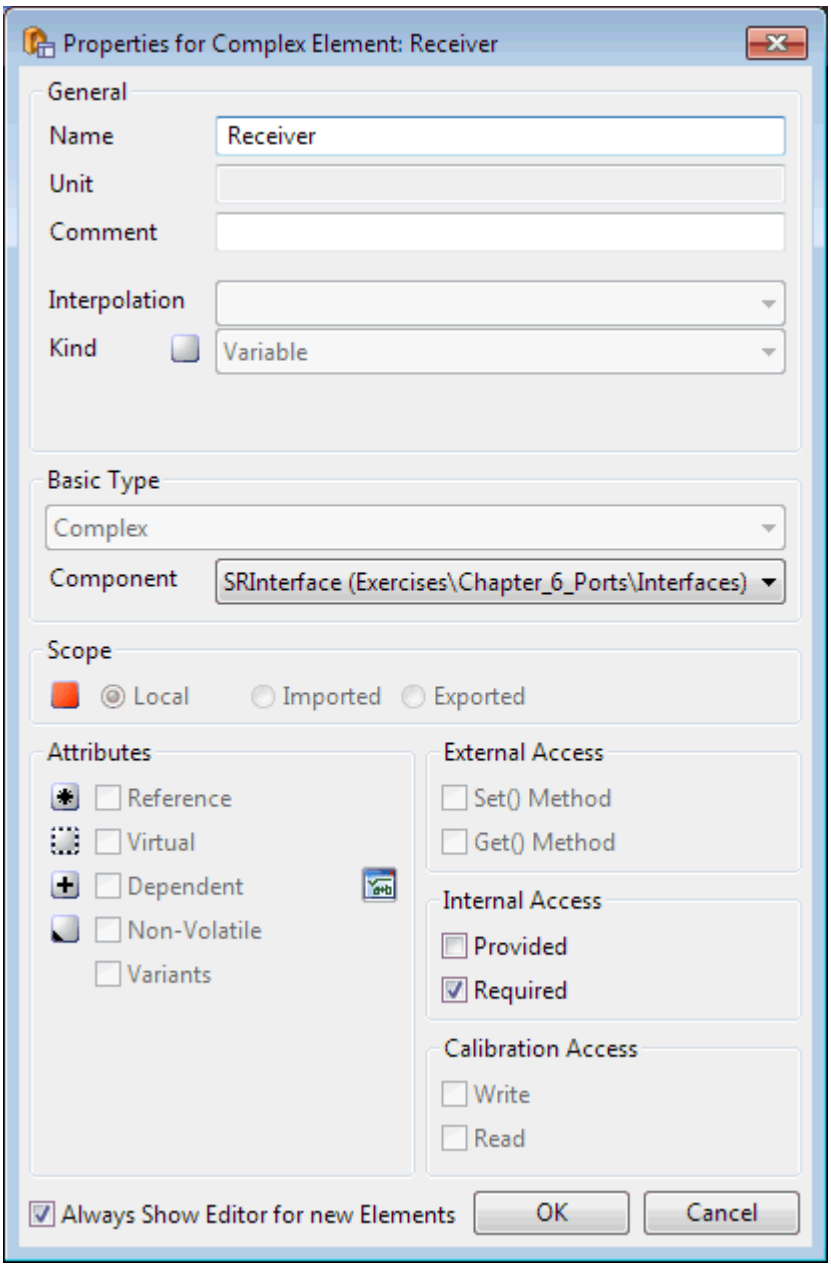


図 33: `SRInterface` 型の R ポート `Receiver`

- エレメント `Receiver::SRInterface` をソフトウェアコンポーネントエディタの“Outline”タブから描画エリアにドラッグ&ドロップします。

エレメント Speed を持つ P ポート Receiver が、描画エリアに以下のように表示されます。

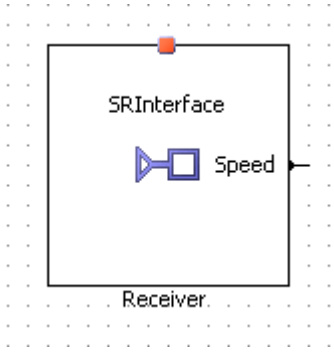


図 34: ソフトウェアコンポーネントエディタの描画エリアに表示された R ポート Receiver

ソフトウェアコンポーネントの型定義内で、R ポートの名前は<SHORT-NAME>エレメントで定義されます。この名前は、このソフトウェアコンポーネント型を参照する他のエレメントの中で使用されます。R ポートのショートネームは有効な C 識別子である必要があります。

R ポートの定義は、<REQUIRED-INTERFACE-TREF>エレメント内でインターフェース定義を参照する必要があります。

```
<R-PORT-PROTOTYPE>
   <SHORT-NAME>Receiver</SHORT-NAME>
   <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
   /ASCET_interfaces/Impl/SRInterface
   </REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

コード 50: ARXML コード – R ポート Receiver の定義（AUTOSAR R3.1.2）

さらに AUTOSAR R4.0.*では、個々のデータエレメントの詳細を含む<PROVIDED-COM-SPECS>エレメントが必要です。以下に詳細情報の一例を示します。

- <DATA-ELEMENT-REF> - データエレメントの識別
- <INIT-VALUE> - データエレメントの初期値

```
<R-PORT-PROTOTYPE>
   <SHORT-NAME>Receiver</SHORT-NAME>
   <REQUIRED-COM-SPECS>
      <NONQUEUED-RECEIVER-COM-SPEC>
         <DATA-ELEMENT-REF DEST="VARIABLE-DATA-PROTOTYPE">
         /ASCET_Interfaces/Impl/SRInterface/Speed</DATA-ELEMENT-REF>
         <INIT-VALUE>
            <NUMERICAL-VALUE-SPECIFICATION>
               <VALUE>0</VALUE>
            </NUMERICAL-VALUE-SPECIFICATION>
         </INIT-VALUE>
      </NONQUEUED-RECEIVER-COM-SPEC>
   </REQUIRED-COM-SPECS>
   <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">
   /ASCET_Interfaces/Impl/SRInterface</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

コード 51: ARXML コード – R ポート Receiver の定義（AUTOSAR R4.0.2）

クライアントポートを作成する:

- “Software Component Editor for: Swc”ウィンドウで **Insert → Component** を選択します。
  “Select item...”ダイアログボックスが開きます。
- “Select item...”ダイアログボックスの“1 Database”または“1 Workspace”ペインでインターフェース `CSInterface` を選択して **OK** をクリックします。
  “Properties for complex element: CSInterface”ダイアログボックスが開きます。
- このポートの名前を `Client` にし、“Internal Access”フィールドで **Required** をオンにして **OK** をクリックします。

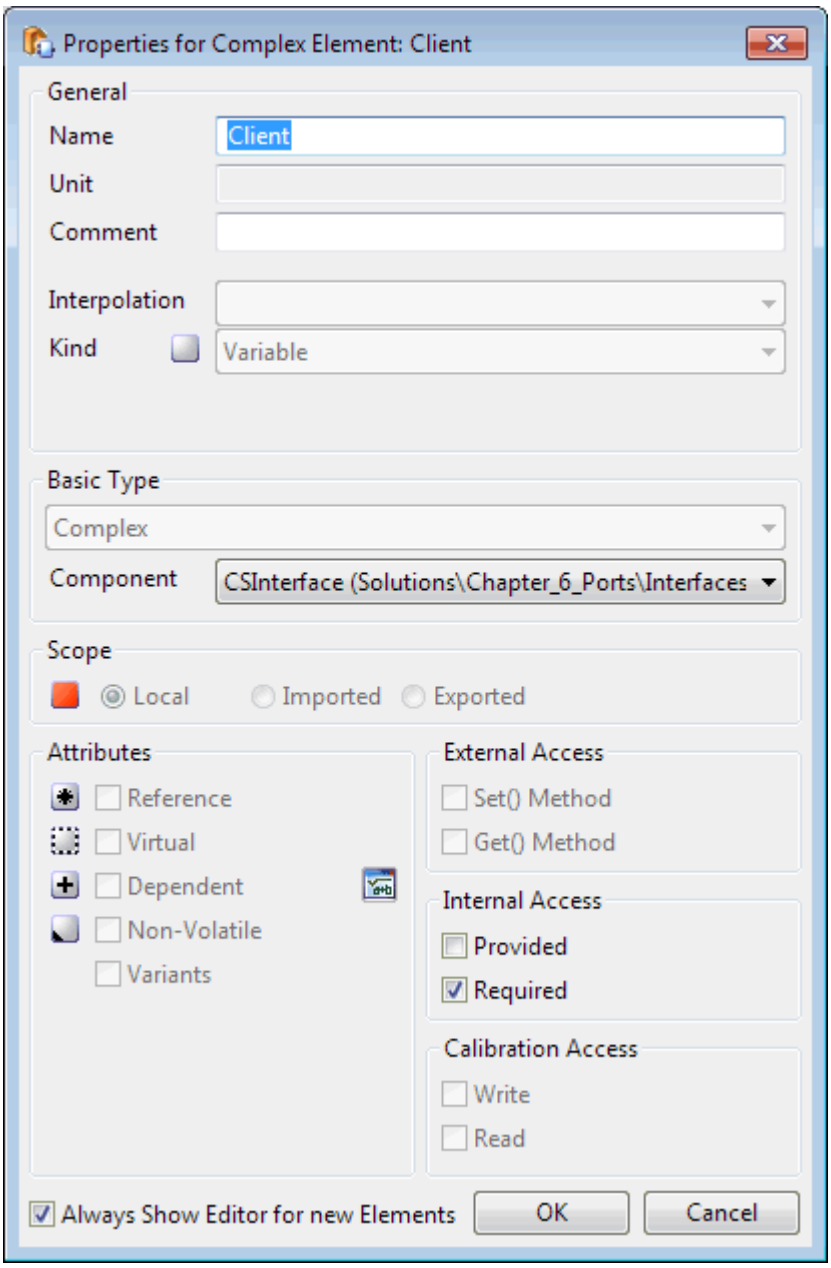


図 35: `CSInterface` 型の R ポート `Client`

- エレメント `Client::CSInterface` をソフトウェアコンポーネントエディタの“Outline”タブから描画エリアにドラッグ&ドロップします。

- 必要に応じて、以下のようにしてインターフェース CSInterface のフレキシブルレイアウトを有効にします。
  - コンポーネントマネージャを選択します。
  - “1 Database”または“1 Workspace”ペインで、CSInterface を右クリックしてショートカットメニューから **Flexible Class Layout → Active** を選択します。

    “Change Flexible Class Layout State”ダイアログボックスで、CSInterface を選択して **OK** をクリックします。
- 描画エリアの Client ポートを右クリックし、ショートカットメニューから **Ports →Methods** を選択します。

  “Port Editor<CSInterface>”ダイアログボックスが開きます。
- メソッド Notification を無効にして **OK** をクリックします。

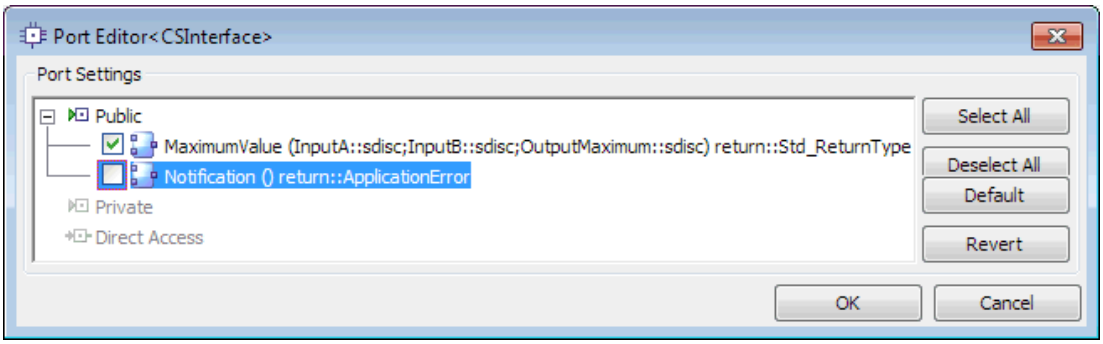


図 36: メソッドの有効／無効を指定するためのポートエディタ

- Client ブロックのサイズとピン位置を調整します。

  オペレーション MaximumValue を持つ R ポート Client が、描画エリアに以下のように表示されます。

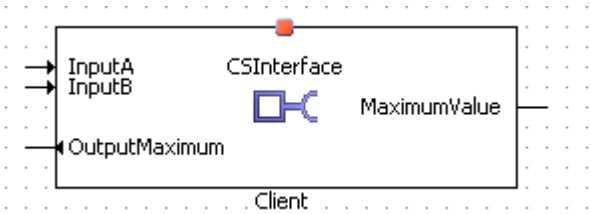


図 37: ソフトウェアコンポーネントエディタの描画エリア内に配置された R ポート Client

R ポートの定義では、<REQUIRED-INTERFACE-TREF>エレメント内でインターフェース定義を参照する必要があります。

```
<R-PORT-PROTOTYPE>
   <SHORT-NAME>Client</SHORT-NAME>
   <REQUIRED-INTERFACE-TREF DEST="CLIENT-SERVER-INTERFACE">
   /ASCET_interfaces/Impl/CSInterface
   </REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

コード 52: ARXML コード – R ポート Client の定義（全 AUTOSAR バージョン）

適合ポートを作成する:

- “Software Component Editor for: Swc”ウィンドウで **Insert → Component** を選択します。

  “Select item...”ダイアログボックスが開きます。

- “Select item...”ダイアログボックスの“1 Database”または“1 Workspace”ペインでインターフェース `CalInterface` を選択して **OK** をクリックします。

  “Properties for complex element: CSInterface”ダイアログボックスが開きます。
- このポートの名前を `Calibration` にして、**OK** をクリックします。

  "Internal Access" は **Required** にセットされていて、変更できません。
- エレメント `Calibration::CalInterface` を "Outline" タブからドラッグし、ソフトウェアコンポーネントエディタの描画エリアにドロップします。

  エレメント `CalParam1`、`CalParam2`、`CalParam3` を持つ R ポート `Receiver` が描画エリアに以下のように表示されます。

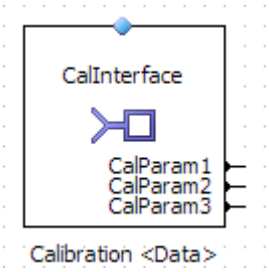


図 38: ソフトウェアコンポーネントエディタの描画エリア内に配置された R ポート `Calibration`

R ポートの定義は、<REQUIRED-INTERFACE-TREF>エレメントによるインターフェース定義を参照する必要があります。

```
<R-PORT-PROTOTYPE>
   <SHORT-NAME>Calibration</SHORT-NAME>
   <REQUIRED-INTERFACE-TREF DEST="CALPRM-INTERFACE">
   /ASCET_interfaces/Impl/CalInterface</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

コード 53: ARXML コード – R ポート `Calibration` の定義（AUTOSAR R3.1.2）

さらに AUTOSAR R4.0.*では、個々のパラメータの詳細情報を定義する<PARAMETER-REQUIRE-COM-SPEC>エレメントを含む<REQUIRED-COM-SPECS>エレメントが 1 つ必要です。以下に詳細情報の一例を示します。

- <PARAMETER-REF> - パラメータの識別
- <INIT-VALUE> - パラメータの初期値

```
<R-PORT-PROTOTYPE>
  <SHORT-NAME>Calibration</SHORT-NAME>
  <REQUIRED-COM-SPECS>
    <PARAMETER-REQUIRE-COM-SPEC>
      <INIT-VALUE>
        <NUMERICAL-VALUE-SPECIFICATION>
          <VALUE>FALSE</VALUE>
        </NUMERICAL-VALUE-SPECIFICATION>
      </INIT-VALUE>
      <PARAMETER-REF DEST="PARAMETER-DATA-PROTOTYPE">
      /ASCET_Interfaces/Impl/CalInterface/CalParam1</PARAMETER-REF>
    </PARAMETER-REQUIRE-COM-SPEC>
    ...
  </REQUIRED-COM-SPECS>
  <REQUIRED-INTERFACE-TREF DEST="PARAMETER-INTERFACE">
  /ASCET_Interfaces/Impl/CalInterface</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

コード 54: ARXML コード – R ポート `Calibration` の定義（AUTOSAR R4.0.2）

NV データポートを作成する:

注記

NV データのインターフェースとポートは、AUTOSAR R4.0.*でのみ使用できます。

- “Software Component Editor for: Swc”ウィンドウで、**Insert → Component** を選択します。
  “Select item...”ダイアログボックスが開きます。
- “Select item...”ダイアログボックスの“1 Database”または“1 Workspace”ペインでインターフェース `NVData_Interface` を選択して **OK** をクリックします。
  “Properties for complex element: NVData_Interface”ダイアログボックスが開きます。
- このポートの名前を `NVData` にして、**OK** をクリックします。
  "Internal Access" は **Required** にセットされていて、変更できません。
- エレメント `NVData::NVData_Interface` を "Outline" タブからドラッグし、ソフトウェアコンポーネントエディタの描画エリアにドロップします。
  エレメント `Speed_NV` を持つ R ポート `NVData` が描画エリアに以下のように表示されます。

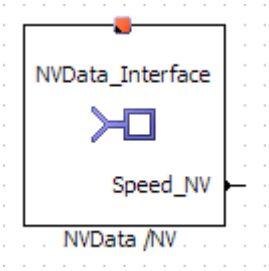


図 39: ソフトウェアコンポーネントエディタの描画エリア内に配置された R ポート `NVData`

さらに AUTOSAR R4.0.*では、個々の NV データエレメントの詳細情報を定義する<NV-REQUIRE-COM-SPEC>エレメントを含む<REQUIRED-COM-SPECS>エレメントが 1 つ必要です。以下に詳細情報の一例を示します。

- <VARIABLE-REF> - NV データエレメントの識別
- <INIT-VALUE> - NV データエレメントの初期値

```
<R-PORT-PROTOTYPE>
  <SHORT-NAME>NVData</SHORT-NAME>
  <REQUIRED-COM-SPECS>
    <NV-REQUIRE-COM-SPEC>
      <INIT-VALUE>
        <NUMERICAL-VALUE-SPECIFICATION>
          <VALUE>0</VALUE>
        </NUMERICAL-VALUE-SPECIFICATION>
      </INIT-VALUE>
      <VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
      /ASCET_Interfaces/Impl/NVData_Interface/Speed_NV</VARIABLE-REF>
    </NV-REQUIRE-COM-SPEC>
  </REQUIRED-COM-SPECS>
  <REQUIRED-INTERFACE-TREF DEST="NV-DATA-INTERFACE">
  /ASCET_Interfaces/Impl/NVData_Interface</REQUIRED-INTERFACE-TREF>
</R-PORT-PROTOTYPE>
```

コード 55: ARXML コード – R ポート NVData の定義（AUTOSAR R4.0.2）

8 内部ビヘイビア（Internal Behavior）

ソフトウェアコンポーネントの内部ビヘイビアは、そのコンポーネントを実装するコードがどのようにポートとのインタラクションを行うかを定義するものです。本章ではこの内部ビヘイビアを設定する方法を説明します。

内部ビヘイビアエレメントは、実行時においてソフトウェアコンポーネントがどのように RTE とインタラクションを行うかを定義するものです。ソフトウェアコンポーネントの内部ビヘイビアは以下のものを定義します。

- ソフトウェアコンポーネントに属するランナブルエンティティ、また各ランナブルとそのソフトウェアコンポーネントのポートとの間にインタラクションがある場合は、そのインタラクションの方法
- 実行時にランナブルエンティティを起動するイベント
- 1 つのソフトウェアコンポーネント内のランナブル間通信に使用されるインターランナブル変数
- 各ランナブルエンティティがそのコードの全体または一部を相互排他的に実行できるようにするための排他的領域

1 つの内部ビヘイビアディスクリプションは 1 つのソフトウェアコンポーネントにのみ適用されるので、適用されるソフトウェアコンポーネント型を参照する必要があります。AUTOSAR R3.1.5 以前では、この参照は<COMPONENT-REF>エレメントを使用して定義し、AUTOSAR R4.0.*では<DATA-TYPE-MAPPING-REF>エレメントを使用して定義します。

```
<INTERNAL-BEHAVIOR>
  <SHORT-NAME>bSwc_IRV</SHORT-NAME>
  <COMPONENT-REF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE">
  /ASCET_swcomponents/Impl/Swc_IRV</COMPONENT-REF>
  <!-- events that cause runnable activation -->
  <EVENTS>
    ...
  </EVENTS>
  <!-- specification of mutual exclusive areas -->
  <EXCLUSIVE-AREAS>
    ...
  </EXCLUSIVE-AREAS>
  <!-- variables for communication between runnables -->
  <INTER-RUNNABLE-VARIABLES>
    ...
  </INTER-RUNNABLE-VARIABLES>
  <!-- runnable entities -->
  <RUNNABLES>
    ...
  </RUNNABLES>
</INTERNAL-BEHAVIOR>
```

コード 56: ARXML コード – SWC の内部ビヘイビアディスクリプション（AUTOSAR R3.1.2）

```
<INTERNAL-BEHAVIORS>
  <SWC-INTERNAL-BEHAVIOR>
    <SHORT-NAME>bSwc</SHORT-NAME>
    <DATA-TYPE-MAPPING-REFS>
      <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">
      /ASCET_Mappings/DataMappings/Impl</DATA-TYPE-MAPPING-REF>
    </DATA-TYPE-MAPPING-REFS>
    <EXCLUSIVE-AREAS>
      ...
    </EXCLUSIVE-AREAS>
    <EVENTS>
      ...
    </EVENTS>
    <EXPLICIT-INTER-RUNNABLE-VARIABLES>
      ...
    </EXPLICIT-INTER-RUNNABLE-VARIABLES>
    <IMPLICIT-INTER-RUNNABLE-VARIABLES>
      ...
    </IMPLICIT-INTER-RUNNABLE-VARIABLES>
    <RUNNABLES>
      ...
    </RUNNABLES>
  </SWC-INTERNAL-BEHAVIOR>
</INTERNAL-BEHAVIORS>
```

コード 57: ARXML コード – SWC の内部ビヘイビアディスクリプション（AUTOSAR R4.0.2）

内部ビヘイビアの名前は<SHORT-NAME>エレメントで定義します。名前は他のエレメントがビヘイビアを参照する際に使用します。ASCET は、ソフトウェアコンポーネントの名前にプレフィックス b を付加したものを自動的に内部ビヘイビアの名前として使用します。

内部ビヘイビアのショートネームは有効な C 識別子である必要はありませんが、強制的に行われる XML スキーマの構文チェックをパスできるものである必要があります。

以下の項では、まずイベントとランナブルエンティティの概略を説明し、イベントとランナブルエンティティの基本的枠組みを概説し、さらにランナブルエンティティとインターフェースとの間で行われる各種インタラクションを実現するための RTE の設定方法を紹介します。

8.1 イベント

イベントは、生成された RTE が実行時にランナブルエンティティをどのようにトリガするかを制御します。ASCET V6.2 は以下のイベントをサポートしています。

- TIMING-EVENT（タイミングイベント）– ランナブルエンティティを周期的に起動します。このイベントを使用してランナブルエンティティを実行させ、R ポートをポーリングしてデータが受信されているかどうかを調べたり、サーバーを呼び出したり（つまりクライアントになったり）、P ポートでデータを送信したり、または単純に内部的なソフトウェアコンポーネント機能を実行したりできます。タイミングイベントに応えて起動されるランナブルエンティティは「タイムトリガードである（"time-triggered"）」と表現されます。
- OPERATION-INVOKED-EVENT （オペレーション呼び出しイベント） – ランナブルエンティティを起動し、クライアント／サーバーインターフェースを使用する P ポートに対するオペレーションを実行するためのサーバーコールを行います。
- MODE-SWITCH-EVENT （モード切り替えイベント）– アプリケーションモードの開始時または終了時のいずれかでランナブルエンティティを起動します。

イベントを定義する構造は、コード 58 とコード 59 のようになります。実際の各イベントのシーケンスは、イベント名で決まります。

```
<EVENTS>
   <OPERATION-INVOKED-EVENT>
      <!-- elements for event/runnable relationship -->
   </OPERATION-INVOKED-EVENT>
   <TIMING-EVENT>
      <!-- elements for event/runnable relationship -->
   </TIMING-EVENT>
   <MODE-SWITCH-EVENT>
      <!-- elements for event/runnable relationship -->
   </MODE-SWITCH-EVENT>
</EVENTS>
```

コード 58: ARXML コード – イベントの定義（AUTOSAR R3.1.2）

```
<EVENTS>
   <OPERATION-INVOKED-EVENT>
      ...
   </OPERATION-INVOKED-EVENT>
   <TIMING-EVENT>
      ...
   </TIMING-EVENT>
   <SWC-MODE-SWITCH-EVENT>
      ...
   </SWC-MODE-SWITCH-EVENT>
</EVENTS>
```

コード 59: ARXML コード – イベントの定義（AUTOSAR R4.0.2）

イベントを使用すると、そのイベントの発生時にランナブルエンティティを起動させることができます。各イベントは、そのイベントの発生時に起動されるランナブルエンティティを参照します。

8.1.1 タイミングイベント

「タイミングイベント」は、ランナブルエンティティが周期的に OS により起動されることを示すものです。RTE ジェネレータはこの情報を使用して適切なスケジュールテーブルを生成します。このテーブルは、アプリケーションコードによって「チック」（カウントアップ）される必要があります。

タイミングイベントを作成する:

- “Software Component Editor for: Swc”ウィンドウの“Event Specification”タブを選択します。
- **Event →Add Event** を選択し、そのイベントの名前を `Cyclic_10ms` にします。
- “Event Kind” コンボボックスから `Timing` を選択します。
- “Period ”フィールドに、0.01 秒を周期として入力します。

  タイミングイベント `Cyclic_10ms` が“Event Specification”タブに次のように表示されます

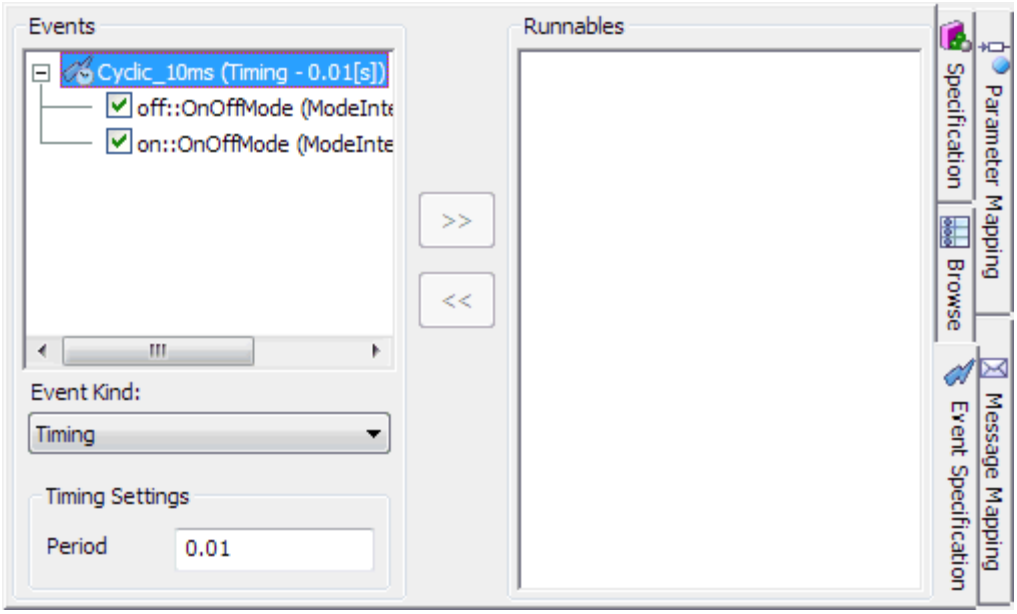


図 40: タイミングイベント Cyclic_10ms の定義

ASCET では、このタイミングイベントがどのモード内で起動されるようにするかを指定することができます。アプリケーションモードの使用については第 9 章「モード」を参照してください。

タイミングイベントがランナブルエンティティにマッピングされると（8.2 項「ランナブルエンティティ」を参照）、ASCET はコンフィギュレーション言語で以下の<TIMING-EVENT>エレメントを生成します。

```
<EVENTS>
   <TIMING-EVENT>
      <SHORT-NAME>Cyclic_10ms</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
      /ASCET_swcomponents/Impl/bSwc/runnable
      </START-ON-EVENT-REF>
      <PERIOD>0.01</PERIOD>
   </TIMING-EVENT>
</EVENTS>
```

コード 60: ARXML コード – タイミングイベントの定義（全 AUTOSAR バージョン）

タイミングイベントの名前は<SHORT-NAME>エレメントで定義されます。この名前は、このタイミングイベントを参照する他のエレメントの中で使用されます。タイミングイベントのショート名は有効な C 識別子である必要はありません。

<START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">エレメントでは、このイベントの発生時に起動されるランナブルエンティティを定義します。<PERIOD>エレメントでは、RTE ジェネレータに使用されるタイムラスタを秒単位で定義します。

8.1.2 オペレーション呼び出しイベント

「オペレーション呼び出しイベント」は、サーバーポート作成時に ASCET ソフトウェアコンポーネント内に自動的に挿入されます（サーバーポートの作成方法については、7.1.1 項「P ポート」を参照してください）。

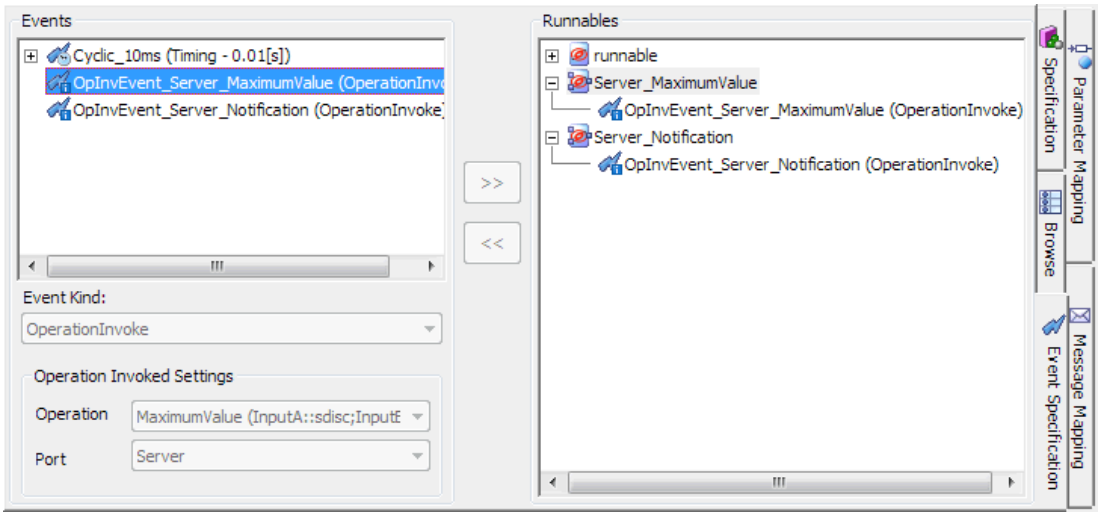


図 41: サーバーオペレーション MaximumVal および Notification のためのオペレーション呼び出しイベント

オペレーション呼び出しイベントは、<OPERATION-INVOKED-EVENT>エレメントで定義されます。各<OPERATION-INVOKED-EVENT>エレメントには以下のものが定義されている必要があります。

- <SHORT-NAME> - このイベントを参照する際に使用する名前（ASCET 上でユーザーがマニュアル操作で編集できます）
- <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY"> - ランナブルエンティティへの参照
- <OPERATION-IREF> - オペレーションプロトタイプおよびサーバーポートへの参照

オペレーション MaximumValue 用のオペレーション呼び出しイベントは、コンフィギュレーション言語で以下のように定義されます。

```
<OPERATION-INVOKED-EVENT>
   <SHORT-NAME>OpInvEvent_Server_MaximumValue</SHORT-NAME>
   <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
   /ASCET_swcomponents/Impl/bSwc/Server_MaximumValue
   </START-ON-EVENT-REF>
   <OPERATION-IREF>
      <P-PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
      /ASCET_swcomponents/Impl/Swc/Server
      </P-PORT-PROTOTYPE-REF>
      <OPERATION-PROTOTYPE-REF DEST="OPERATION-PROTOTYPE">
      /ASCET_interfaces/Impl/CSInterface/MaximumValue
      </OPERATION-PROTOTYPE-REF>
   </OPERATION-IREF>
</OPERATION-INVOKED-EVENT>
```

コード 61: ARXML コード – オペレーション呼び出しイベントの定義（AUTOSAR R3.1.2）

```
<OPERATION-INVOKED-EVENT>
   <SHORT-NAME>OpInvEvent_Server_MaximumValue</SHORT-NAME>
   <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
   /ASCET_ComponentTypes/SWC/bSWC/Server_MaximumValue
   </START-ON-EVENT-REF>
   <OPERATION-IREF>
      <CONTEXT-P-PORT-REF DEST="P-PORT-PROTOTYPE">
      /ASCET_ComponentTypes/SWC/Server</CONTEXT-P-PORT-REF>
      <TARGET-PROVIDED-OPERATION-REF DEST="CLIENT-SERVER-OPERATION">
      /ASCET_Interfaces/Impl/CSInterface/MaximumValue
      </TARGET-PROVIDED-OPERATION-REF>
   </OPERATION-IREF>
</OPERATION-INVOKED-EVENT>
```

コード 62: ARXML コード – オペレーション呼び出しイベントの定義（AUTOSAR R4.0.2）

8.1.3 モード切り替えイベント

「モード切り替えイベント」は、アプリケーションモードの開始時または終了時でランナブルエンティティを起動します。

モード切り替えイベントを作成する:

- “Software Component Editor for: Swc”ウィンドウの“Event Specification” タブを選択します。
- **Event →Add Event** を選択し、そのイベントの名前を `ModeEvent` にします。
- **Event Kind** コンボボックスで `ModeSwitch` を選択します。
- モード切り替えについて以下のように設定します。
  - **Activation**: `entry`
  - **Assigned Mode**: `on::OnOffMode`

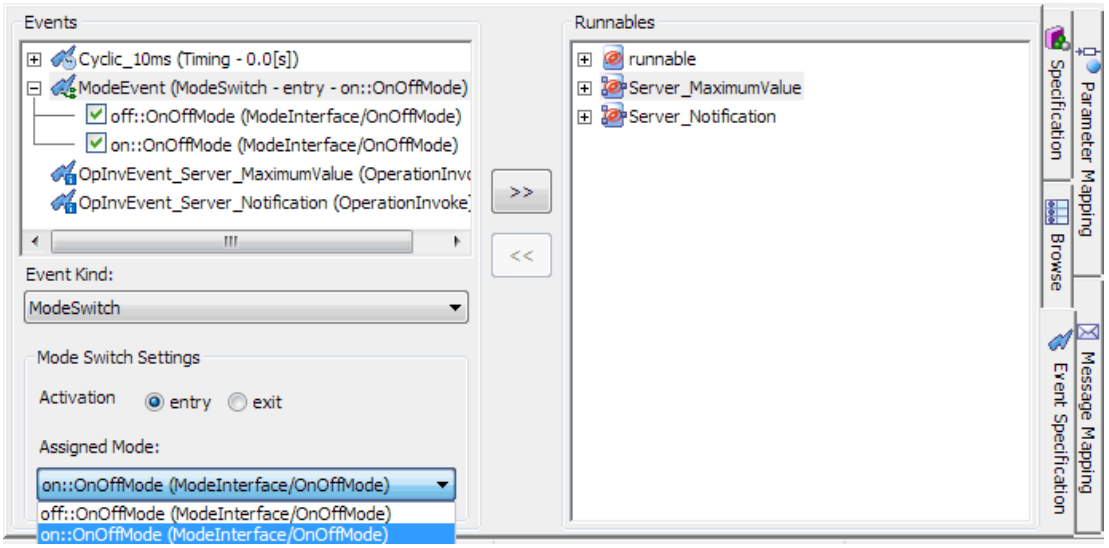


図 42: アプリケーションモード `OnOffMode` のモード on の開始時の `ModeEvent` をモデリングする

このモードイベントがランナブルエンティティにマッピングされると（8.3 項「タイミングイベントへの応答」を参照）、ASCETはコンフィギュレーション言語で以下の<MODE-SWITCH-EVENT>[1] / <SWC-MODE-SWITCH-EVENT>[2]エレメントを生成します。

```
<MODE-SWITCH-EVENT>
   <SHORT-NAME>ModeEvent</SHORT-NAME>
   <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
   /ASCET_swcomponents/Impl/bSwc/ModeRunnable
   </START-ON-EVENT-REF>
   <ACTIVATION>ENTRY</ACTIVATION>
   <MODE-IREF>
      <R-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
      /ASCET_swcomponents/Impl/Swc/ModeInterface
      </R-PORT-PROTOTYPE-REF>
      <MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST=
      "MODE-DECLARATION-GROUP-PROTOTYPE">
      /ASCET_interfaces/Impl/ModeInterface/OnOffMode
      </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
      <MODE-DECLARATION-REF DEST="MODE-DECLARATION">
      /ASCET_types/OnOffMode/on</MODE-DECLARATION-REF>
   </MODE-IREF>
</MODE-SWITCH-EVENT>
```

コード 63: ARXML コード – モード切り替えイベントの定義（AUTOSAR R3.1.2）

```
<SWC-MODE-SWITCH-EVENT>
   <SHORT-NAME>ModeEvent</SHORT-NAME>
   <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
   /ASCET_ComponentTypes/SWC/bSWC/ModeRunnable
   </START-ON-EVENT-REF>
   <ACTIVATION>ON-ENTRY</ACTIVATION>
   <MODE-IREFS>
      <MODE-IREF>
         <CONTEXT-PORT-REF DEST="R-PORT-PROTOTYPE">
         /ASCET_ComponentTypes/SWC/ModeInterface
         </CONTEXT-PORT-REF>
         <CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST=
         "MODE-DECLARATION-GROUP-PROTOTYPE">
         /ASCET_Interfaces/Impl/R4_ModeInterface/OnOffMode
         </CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF>
         <TARGET-MODE-DECLARATION-REF DEST="MODE-DECLARATION">
         /ASCET_Types/ApplicationDataTypes/OnOffMode/on
         </TARGET-MODE-DECLARATION-REF>
      </MODE-IREF>
   </MODE-IREFS>
</SWC-MODE-SWITCH-EVENT>
```

コード 64: ARXML コード – モード切り替えイベントの定義（AUTOSAR R4.0.2）

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

"Events" フィールドには、割り当てられたモードグループ内のすべてのモードがモード切り替えイベントの下に表示されます。これらは個々に有効／無効にできます。1 つ以上のモードが無効になっていると（138 ページの図 57 参照）、コンフィギュレーション言語に<MODE-DEPENDENCY>[1] / <DISABLED-MODE-IREFS>[2] エレメントが追加され、無効化されたモードごとに 1 つずつ <DEPENDENT-ON-MODE-IREF>[1] / <DISABLED-MODE-IREF>[2] が含まれます。

ARXML コードの例は、139 ページのコード 94（AUTOSAR R3.1.2）または 140 ページのコード 95（AUTOSAR R4.0.2）を参照してください。

無効化モードについての詳細は、138 ページの 9.3.3 項「モードの無効化」を参照してください。モード切り替えイベントの名前は<SHORT-NAME>エレメントで定義されます。この名前は、このモード切り替えイベントを参照する他のエレメントの中で使用されます。モード切り替えイベントのショート名は有効な C 識別子である必要はありません。

各<MODE-SWITCH-EVENT>エレメントには以下のものを定義する必要があります。

- <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY"> - ランナブルエンティティへの参照
- <ACTIVATION>（ENTRY または EXIT） - 起動の種類を表す値
- <MODE-IREF> - イベントに割り当てられたモードを定義するエレメント
- <MODE-DEPENDENCY> - モード宣言への参照（必要な場合）

8.2 ランナブルエンティティ

「ランナブルエンティティ」（または単に「ランナブル」とも呼ばれます）は、実行時に RTE によりトリガされるソフトウェアコンポーネント内のコード部分です。ソフトウェアコンポーネントは 1 つまたは複数のランナブルからなり、各ランナブルには実行時に RTE からアクセスできるように独自のハンドルがなければなりません。

ランナブルエンティティを作成する:

- “Software Component Editor for: Swc”ウィンドウの“Outline”タブで、ダイアグラム（例: Main）を選択します。
- **Insert → Runnable** を選択し、そのランナブルの名前を RunnableEntity にします。

すべてのランナブルエンティティは、<INTERNAL-BEHAVIOR>[1] / <SWC-INTERNAL-BEHAVIOR>[2] 内の<RUNNABLES>内に、ソフトウェアコンポーネントテンプレートとして定義されている必要があります。

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SYMBOL>Swc_Impl_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 65: ARXML コード – ランナブルエンティティの定義（AUTOSAR R3.1.2）

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <SYMBOL>SWC_Impl_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 66: ARXML コード – ランナブルエンティティの定義（AUTOSAR R4.0.2）

<RUNNABLE-ENTITY>の名前は、<SHORT-NAME>エレメントで定義されます。この名前は、このランナブルエンティティを参照する他のエレメントの中で使用されます。

<SHORT-NAME>は XML ネームスペースにおけるこのランナブルエンティティの名前を表しますが、これはユーザーコード内の、呼び出される関数ボディの名前を RTE に知らせるものではありません。この情報は、ユーザーのインプリメンテーションで使用される C 関数名にランナブルエンティティをリンクさせる<SYMBOL>宣言により提供されます。この<SYMBOL>名は有効な C 識別子である必要があります。

AUTOSAR R4.0.*では、生成されたコードのメモリクラスは<SW-ADDR-METHOD-REF>エレメントにより決定されます。

ランナブルエンティティのシンボルは ASCET では必須の情報ではありません。これが定義されない場合は、ASCET が生成する C コード内の、ランナブルエンティティを実装する関数の名前が採用されます。上記の例では、ASCET は C 関数 SWC_IMPL_RunnableEntity を生成します。シンボルが定義されている場合、ASCET はそのシンボルに従ってランナブルエンティティの C コードを生成します。

ランナブル用の C 識別子を設定する:

- ソフトウェアコンポーネントエディタの“Outline”タブで、ランナブル RunnableEntity を選択し、**Edit → Implementation** を選択します。
  “Implementation for: RunnableEntity”ダイアログボックスが開きます。
- シンボル RteRunnable_Swc_RunnableEntity を入力します。
- **OK** をクリックします。

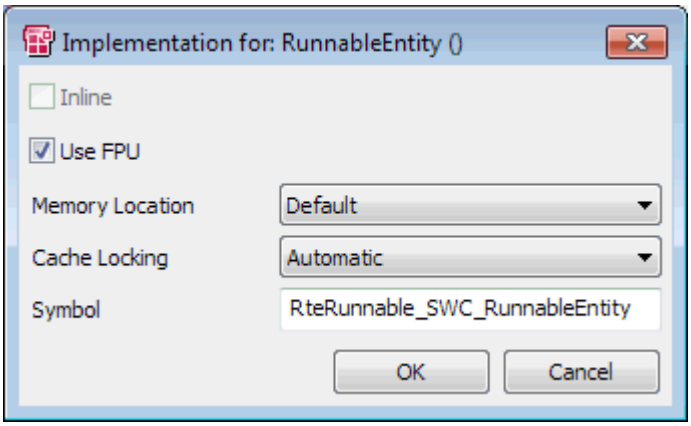


図 43: ランナブル RunnableEntity 用のシンボル RteRunnable_Swc_RunnableEntity を設定する

これにより、ASCET は実装されるランナブルの C コードを生成し、その名前を RteRunnable_Swc_RunnableEntity にします。この例ではファイル Swc.c を参照してください。

```
FUNC(void, CODE) RteRunnable_SWC_RunnableEntity (void)
{
...
}
```

<RUNNABLE-ENTITY> の定義は以下のようになります。

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SYMBOL>RteRunnable_SWC_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 67: ARXML コード – ユーザー定義された<SYMBOL>によるランナブルエンティティの定義（AUTOSAR R3.1.2）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 68: ARXML コード – ユーザー定義された<SYMBOL>によるランナブルエンティティの定義（AUTOSAR R4.0.2）

ソフトウェアコンポーネントのポートとのインタラクションを必要としないランナブルエンティティの場合は、上記のような宣言で十分です。しかし、ポートによる通信が必要なランナブルエンティティの場合は、インタラクションを行う API を RTE ジェネレータが生成できるようにするため、以下の例のような情報を追加する必要があります。

1. ランナブルエンティティが送信できるデータアイテム
2. ランナブルエンティティが受信できるデータアイテム
3. ランナブルエンティティが呼び出すサーバー、および結果の返されかた

同じ 1 つのランナブルエンティティを使用して、1 つのポートでデータを受信した後に別のポートでデータを送信したり、1 つのポートでデータを受信してから受信データを処理するサーバーポートを呼び出したりすることができます。たとえば、R ポートから整数値を読み取り、それに 2 を掛け、その結果を P ポートから送信するランナブルエンティティを作成することができます。

オペレーション呼び出しイベントから呼び出されないランナブルエンティティでは、最小起動インターバルを指定して起動の間隔を制御することもできます。この機能は、最小起動インターバルより短い間隔でランナブルの起動要求がかかった場合に、要求された起動を遅らせ、所定のインターバル内にランナブルが 2 回以上起動されないようにするものです。

注記

最小起動インターバルを使用する場合は、使用する RTE ジェネレータにそのランナブルの起動がどのように実装されているかを必ず確認してください。

8.3 タイミングイベントへの応答

タイミングイベントに関連付けられたランナブルエンティティは、実行時に周期的に実行されます。タイミングイベントによりランナブルエンティティの実行頻度が決まります。

<TIMING-EVENT>エレメントで発生の周期<PERIOD>を秒単位で指定し、<START-ON-EVENT-REF>エレメントで、コンポーネントの内部ビヘイビアに定義されているランナブルエンティティを参照します。周期をゼロにすることはできません。

次の例では、所定のランナブルエンティティが 10 マイクロ秒ごとに起動されるように RTE を設定する方法を紹介します。

ランナブルにタイミングイベントを割り当てる:

- “Software Component Editor for: Swc”ウィンドウの“Event Specification”タブでを選択します。
- “Events”フィールドで、イベント `Cyclic_10ms` を選択します。
- “Runnables”フィールドで、ランナブル `RunnableEntity` を選択します。
- **Event →Assign Event** を選択するか、**>>**ボタンをクリックします。

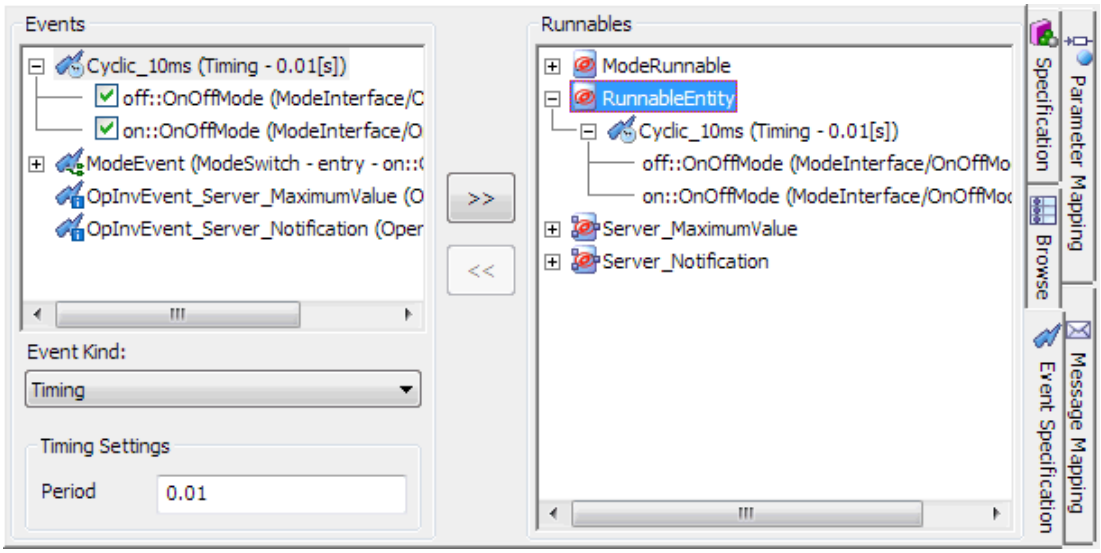


図 44: イベント `Cyclic_10ms` を `RunnableEntity` に割り当てる

<TIMING-EVENT>エレメント内の<START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">にイベント発生時に起動されるランナブルエンティティが定義され、<PERIOD>に RTE ジェネレータが使用する周期が定義されます。

タイミングイベントの名前は<SHORT-NAME>エレメントで定義されます。この名前は、このタイミングイベントを参照する他のエレメントの中で使用されます。タイミングイベントのショート名は有効な C 識別子である必要はありません。

103 ページのコード 60: ARXML コード – タイミングイベントの定義（全 AUTOSAR バージョン）も参照してください。

8.4 ポートへの送信

センダ／レシーバインターフェースを提供するソフトウェアコンポーネントには、そのインターフェースを介してデータを送信するランナブルエンティティを 1 つ以上定義する必要があります。

ランナブルは以下の 2 通りの方法でデータを送信できます。

- 明示的な送信 ― RTE は明示的な API コールを生成します。この呼び出しは最適化機能によってマクロ化することができ、送信データのキューイングの有無も指定できます。

- 暗黙的な送信 ― RTE は暗黙的な API コールを生成します。この呼び出しも最適化機能によってマクロ化することができますが、送信データのキューイングは行えません。

センダにとってはランナブルエンティティがどのようにトリガされるかは問題ではないので、ランナブルエンティティの起動にはどのイベントでも使用できます。

8.4.1 明示的な送信

ポートへの明示的な送信を行う:

- 87 ページの「センダポートを作成する」の方法で、P ポート Sender を作成します。
- P ポート Sender を"Outline"タブからドラッグし、ソフトウェアコンポーネントエディタの描画エリアにドロップします。
- **RTE Access** ボタンを使用して、RTE アクセス演算子を作成し、描画エリアに配置します。
- RTE アクセス演算子の出力を Sender ポートのデータエレメント Speed に接続します。
- "Outline" タブで、ランナブル RunnableEntity を選択し、Speed のシーケンスコールをダブルクリックします。

  ASCET はランナブルエンティティ RunnableEntity 内でデータエレメント Speed が送信されるように、シーケンス番号（ここでは 5）を自動的に割り当てます。
- 値が 120 のリテラルをダイアグラムに追加し、このリテラルを RTE アクセス演算子の入力に接続します。

  これでコード生成が行えるようになりました。コード生成の方法は 27 ページの「プロジェクトのコードを生成する」を参照してください。

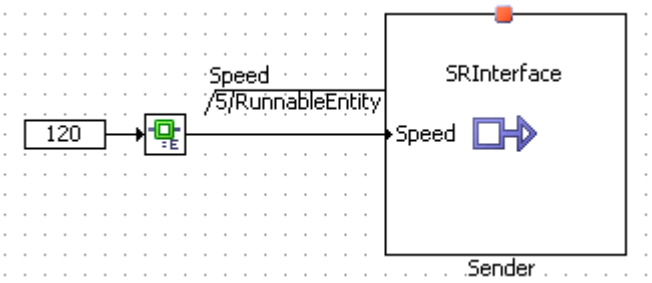


図 45: 値 120 をセンダポートに明示的に送信する

データを明示的に送信するランナブルエンティティには、<DATA-SEND-POINTS>エレメントで所定のインターフェースに送信するデータアイテムを定義する必要があります。

AUTOSAR R3.1.5 以前では、送信データアイテムは<DATA-SEND-POINT>エレメント内で定義します。各<DATA-SEND-POINT>エレメントには以下のプロパティを定義する必要があります。

- <SHORT-NAME> - アイテム参照用の名前（有効な C 識別子である必要はありません）
- <DATA-ELEMENT-IREF> - 以下のプロパティを 1 つずつ含むエレメント
  - <P-PORT-PROTOTYPE-REF>（P ポートへの参照）
  - <DATA-ELEMENT-PROTOTYPE-REF>（送信エレメントへの参照）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-SEND-POINTS>
      <DATA-SEND-POINT>
         <SHORT-NAME>dataSendPoint1</SHORT-NAME>
         <DATA-ELEMENT-IREF>
            <P-PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
            /ASCET_swcomponents/Impl/Swc/Sender</P-PORT-PROTOTYPE-REF>
            <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-PROTOTYPE">
            /ASCET_interfaces/Impl/SRInterface/Speed
            </DATA-ELEMENT-PROTOTYPE-REF>
         </DATA-ELEMENT-IREF>
      </DATA-SEND-POINT>
   </DATA-SEND-POINTS>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SYMBOL>RteRunnable_SWC_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 69: ARXML コード – 明示的送信を行うランナブルエンティティ（AUTOSAR R3.1.2）

AUTOSAR R4.0.*では、送信データアイテムは<VARIABLE-ACCESS> エレメントで指定されます。各<VARIABLE-ACCESS>には以下のプロパティを定義する必要があります。

- <SHORT-NAME> - アイテム参照用の名前（有効な C 識別子である必要はありません）
- <ACCESSED-VARIABLE> - 以下のプロパティを 1 つずつ含む<AUTOSAR-VARIABLE-IREF>エレメントを含むエレメント
  - <P-PORT-PROTOTYPE-REF>（P ポートへの参照）
  - <TARGET-DATA-PROTOTYPE-REF>（送信エレメントへの参照）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-SEND-POINTS>
      <VARIABLE-ACCESS>
         <SHORT-NAME>DataSendPoint1</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <AUTOSAR-VARIABLE-IREF>
               <PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
                   /ASCET_ComponentTypes/SWC/Sender</PORT-PROTOTYPE-REF>
               <TARGET-DATA-PROTOTYPE-REF DEST="DATA-PROTOTYPE">
                   /ASCET_Interfaces/Impl/SRInterface/Speed
                   </TARGET-DATA-PROTOTYPE-REF>
            </AUTOSAR-VARIABLE-IREF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
   </DATA-SEND-POINTS>
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 70: ARXML コード – 明示的送信を行うランナブルエンティティ（AUTOSAR R4.0.2）

センダにとっては、ランナブルエンティティがどのようにトリガされるかは問題ではないので、ランナブルエンティティの起動にはどのイベントでも使用できます。

ASCET が生成するコードについては 145 ページの 10.3.1 項「ポートへの送信」を参照してください。

8.4.2 暗黙的な送信

ランナブルエンティティは、暗黙的なデータ送信を行うこともできます。このような設定は、C 関数の呼び出しではなく、RTE に定義されているグローバル変数にアクセスするシンプルなマクロとして実装されます。

ASCET で暗黙的な送信をモデリングするには以下の 2 通りの方法があります。

1. RTE アクセスを Explicit から Implicit に変更する
2. RTE アクセス演算子を使用せずに暗黙的通信をモデリングする

RTE アクセスを Implicit に変更する:

- 図 46 のように、描画エリアで、8.4.1 項の例の RTE アクセス演算子を右クリックし、ショートカットメニューから **Access → Implicit** を選択します。

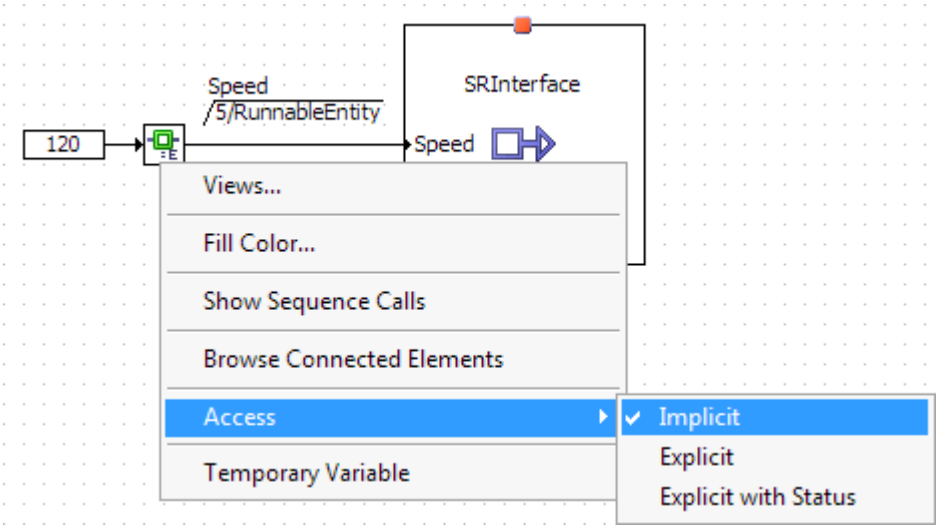


図 46: RTE アクセス演算子のアクセスタイプを Implicit に変更する

ポートへの暗黙的な送信を行う:

- P ポート `Sender` を"Outline"タブからドラッグし、ソフトウェアコンポーネントエディタの描画エリアにドロップします。
- 値が `120` のリテラルを追加し、このリテラルを P ポート `Sender` のデータエレメント `Speed` に接続します。
- "Outline" タブで、ランナブル `RunnableEntity` を選択し、`Speed` のシーケンスコールをダブルクリックします。

  ASCET はランナブルエンティティ `RunnableEntity` 内でデータエレメント `Speed` が送信されるように、シーケンス番号（ここでは 10）を自動的に割り当てます。

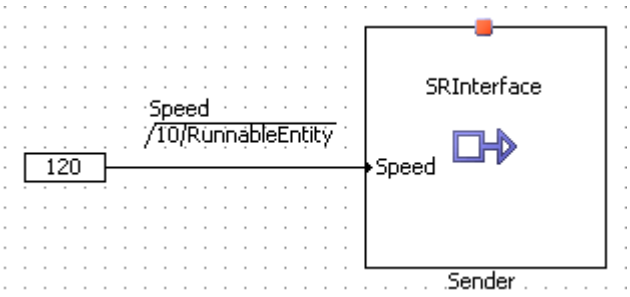


図 47: 値 120 をセンダポートに暗黙的に送信する

暗黙的な送信の設定は明示的な送信の場合とほとんど同じですが、暗黙的な送信の定義には<DATA-SEND-POINTS>エレメントの代わりに<DATA-WRITE-ACCESSS>エレメントが使用されます。

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-WRITE-ACCESSS>
      <DATA-WRITE-ACCESS>
         <SHORT-NAME>dataWriteAccess1</SHORT-NAME>
         <DATA-ELEMENT-IREF>
            <P-PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
            /ASCET_swcomponents/Impl/Swc/Sender</P-PORT-PROTOTYPE-REF>
            <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-PROTOTYPE">
            /ASCET_interfaces/Impl/SRInterface/Speed
            </DATA-ELEMENT-PROTOTYPE-REF>
         </DATA-ELEMENT-IREF>
      </DATA-WRITE-ACCESS>
   </DATA-WRITE-ACCESSS>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SYMBOL>RteRunnable_SWC_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 71: ARXML コード – 暗黙的送信を行うランナブルエンティティ（AUTOSAR R3.1.2）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-WRITE-ACCESSS>
      <VARIABLE-ACCESS>
         <SHORT-NAME>DataWriteAccess1</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <AUTOSAR-VARIABLE-IREF>
               <PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
                   /ASCET_ComponentTypes/SWC/Sender</PORT-PROTOTYPE-REF>
               <TARGET-DATA-PROTOTYPE-REF DEST="DATA-PROTOTYPE">
                   /ASCET_Interfaces/Impl/SRInterface/Speed
                   </TARGET-DATA-PROTOTYPE-REF>
            </AUTOSAR-VARIABLE-IREF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
   </DATA-WRITE-ACCESSS>
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 72: ARXML コード –暗黙的送信を行うランナブルエンティティ（AUTOSAR R4.0.2）

ASCET が生成するコードについては 145 ページの 10.3.1 項「ポートへの送信」を参照してください。

8.5 ポートからの受信

センダ／レシーバインターフェースを必要とするソフトウェアコンポーネントにも、そのインターフェースを介してデータを受信するランナブルエンティティを 1 つ以上定義する必要があります。データは以下の 2 通りの方法で受信できます。

- 暗黙的な受信 – ランナブルはタイミングイベントなどのイベントにより起動され、データを受信する RTE API を呼び出します。
- 明示的な受信 – ランナブルはイベントにより起動され、データを受信する RTE API を呼び出します。レシーバはノンブロッキング API を使用してデータをポーリングします。

8.5.1 明示的な受信

ポートからの明示的な受信を行う:

- 93 ページの「センダポートを作成するセンダポートを作成するセンダポートを作成する」の方法で、R ポート SReceiver を作成します。
- R ポート Receiver を"Outline"タブからドラッグし、ソフトウェアコンポーネントエディタの描画エリアにドロップします。
- **RTE Access** ボタンを使用し、RTE アクセス演算子を作成して描画エリアに配置します。
- Receiver ポートのデータエレメント Speed を RTE アクセス演算子の入力に接続します。
- 符号付き離散変数を挿入し、その名前を SpeedSwc にします。
- 変数 SpeedSwc を、範囲が[-32768, 32767]の sint16 として実装します。
- "Outline" タブで、ランナブル RunnableEntity を選択し、変数 SpeedSwc の空のシーケンスコールをダブルクリックします。

ASCET は SpeedSwc にランナブルエンティティ RunnableEntity のシーケンス番号（ここでは 10）を自動的に割り当てます。

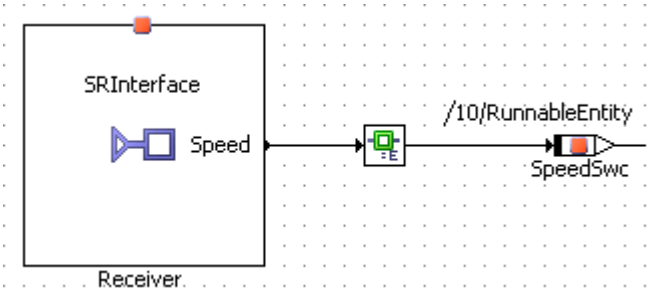


図 48: R ポート Receiver から値 Speed を明示的に受信する

明示的にデータを受信するランナブルには、受信するデータアイテムを<DATA-RECEIVE-POINTS>[1] / <DATA-RECEIVE-POINT-BY-VALUES>[2]エレメント内に定義します。

AUTOSAR R3.1.5 以前では、受信データアイテムは<DATA-RECEIVE-POINT>エレメント内で定義します。各<DATA- RECEIVE -POINT>エレメントには以下のプロパティを定義する必要があります。

- <SHORT-NAME> - アイテム参照用の名前（有効な C 識別子である必要はありません）
- <DATA-ELEMENT-IREF> - 以下のプロパティを 1 つずつ含むエレメント
  - <P-PORT-PROTOTYPE-REF>（P ポートへの参照）
  - <DATA-ELEMENT-PROTOTYPE-REF>（送信エレメントへの参照）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-RECEIVE-POINTS>
      <DATA-RECEIVE-POINT>
         <SHORT-NAME>dataReceivePoint1</SHORT-NAME>
         <DATA-ELEMENT-IREF>
            <R-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
            /ASCET_swcomponents/Impl/Swc/Receiver</R-PORT-PROTOTYPE-REF>
            <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-PROTOTYPE">
            /ASCET_interfaces/Impl/SRInterface/Speed
            </DATA-ELEMENT-PROTOTYPE-REF>
         </DATA-ELEMENT-IREF>
      </DATA-RECEIVE-POINT>
   </DATA-RECEIVE-POINTS>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SYMBOL>RteRunnable_SWC_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 73: ARXML コード – 明示的受信を行うランナブルエンティティ（AUTOSAR R3.1.2）

AUTOSAR R4.0.*では、受信データアイテムは<VARIABLE-ACCESS> エレメントで指定されます。各<VARIABLE-ACCESS>には以下のプロパティを定義する必要があります。

- <SHORT-NAME> - アイテム参照用の名前（有効な C 識別子である必要はありません）
- <ACCESSED-VARIABLE> - 以下のプロパティを 1 つずつ含む<AUTOSAR-VARIABLE-IREF>エレメントを含むエレメント

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

– <P-PORT-PROTOTYPE-REF>（R ポートへの参照）
– <TARGET-DATA-PROTOTYPE-REF>（受信エレメントへの参照）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-RECEIVE-POINT-BY-VALUES>
      <VARIABLE-ACCESS>
         <SHORT-NAME>DataReceivePoint1</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <AUTOSAR-VARIABLE-IREF>
               <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
                   /ASCET_ComponentTypes/SWC/Receiver</PORT-PROTOTYPE-REF>
               <TARGET-DATA-PROTOTYPE-REF DEST="DATA-PROTOTYPE">
                   /ASCET_Interfaces/Impl/SRInterface/Speed
                   </TARGET-DATA-PROTOTYPE-REF>
            </AUTOSAR-VARIABLE-IREF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
   </DATA-RECEIVE-POINT-BY-VALUES>
   ...
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 74: ARXML コード –明示的送信を行うランナブルエンティティ（AUTOSAR R4.0.2）

データ受信を行うランナブルエンティティは、Rポートにポーリングして所定のデータアイテムの有無を調べます。したがって<DATA-RECEIVE-POINTS>[1] / <DATA-RECEIVE-POINT-BY-VALUES>[2]が定義されているランナブルエンティティは、通常、ポーリング周期を規定する<TIMING-EVENT>により起動されます。

ASCET が生成するコードについては 145 ページの 10.3.2 項「ポートからの受信」を参照してください。

8.5.2 暗黙的な受信

ASCET で暗黙的な受信をモデリングするには以下の 2 通りの方法があります

1. RTE アクセスを Explicit から Implicit に変更する
2. RTE アクセス演算子を使用せずに暗黙的な受信をモデリングする

RTE アクセスを Implicit に変更する:

- 図 49 のように、描画領域において、8.5.1 項の例で配置した RTE アクセス演算子を右クリックし、ショートカットメニューから **Access → Implicit** を選択します。

[1] AUTOSAR R3.1.5 以前
[2] AUTOSAR R4.0.*

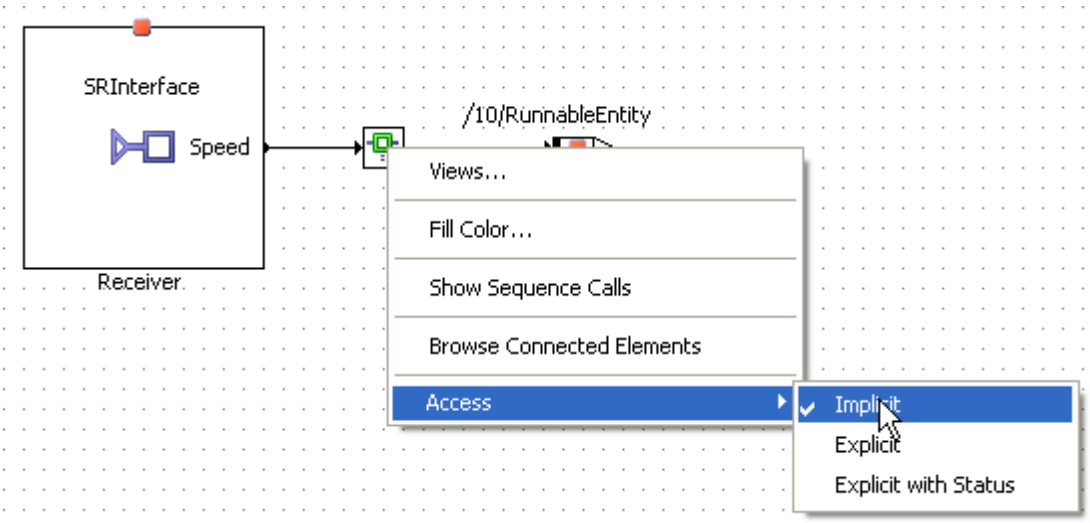


図 49: RTE アクセス演算子のアクセスを Implicit に変更する

ポートからの暗黙的な受信を行う:

- R ポート `Receiver` を"Outline"タブからドラッグし、ソフトウェアコンポーネントエディタの描画エリアにドロップします。
- 符号付き離散変数を挿入して名前を `SpeedSwc` にし、範囲が`[-32768, 32767]`の `sint16` として実装します。
- `Receiver` ポートのデータエレメント `Speed` を変数 `SpeedSwc` に接続します。
- "Outline" タブで、ランナブル `RunnableEntity` を選択し、変数 `SpeedSwc` の空のシーケンスコールをダブルクリックします。

  ASCET は `SpeedSwc` にランナブルエンティティ `RunnableEntity` のシーケンス番号（ここでは `10`）を自動的に割り当てます。

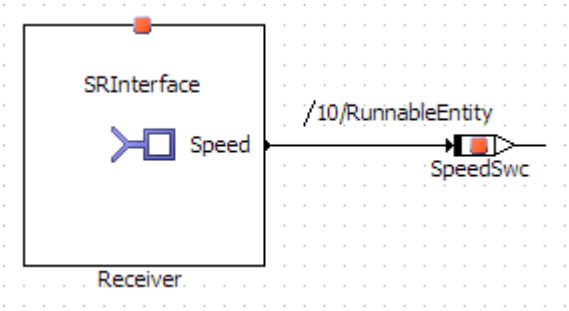


図 50: R ポート `Receiver` から値 `Speed` を暗黙的に受信する

暗黙的にデータを受信するランナブルには、受信するデータアイテムを<DATA-READ-ACCESS>エレメント内に定義します。

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-READ-ACCESSS>
      <DATA-READ-ACCESS>
         <SHORT-NAME>dataReadAccess1</SHORT-NAME>
         <DATA-ELEMENT-IREF>
            <R-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
            /ASCET_swcomponents/Impl/Swc/Receiver</R-PORT-PROTOTYPE-REF>
            <DATA-ELEMENT-PROTOTYPE-REF DEST="DATA-ELEMENT-PROTOTYPE">
            /ASCET_interfaces/Impl/SRInterface/Speed
            </DATA-ELEMENT-PROTOTYPE-REF>
         </DATA-ELEMENT-IREF>
      </DATA-READ-ACCESS>
   </DATA-READ-ACCESSS>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SYMBOL>RteRunnable_SWC_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 75: ARXML コード – 暗黙的受信を行うランナブルエンティティ（AUTOSAR R3.1.2）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <DATA-READ-ACCESSS>
      <VARIABLE-ACCESS>
         <SHORT-NAME>DataReadAccess1</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <AUTOSAR-VARIABLE-IREF>
               <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
                   /ASCET_ComponentTypes/SWC/Receiver</PORT-PROTOTYPE-REF>
               <TARGET-DATA-PROTOTYPE-REF DEST="DATA-PROTOTYPE">
                   /ASCET_Interfaces/Impl/SRInterface/Speed
                   </TARGET-DATA-PROTOTYPE-REF>
            </AUTOSAR-VARIABLE-IREF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
   </DATA-READ-ACCESSS>
   ...
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 76: ARXML コード – 暗黙的受信を行うランナブルエンティティ（AUTOSAR R4.0.2）

各受信データは、<DATA-READ-ACCESS>[1] / <VARIABLE-ACCESS>[2] エレメントで定義します。<DATA-READ-ACCESS>/<VARIABLE-ACCESS>エレメントの名前は<SHORT-NAME>エレメントで定義する必要があり、この名前は他のエレメントがデータ読取りアクセスへの参照を行う際に使用されます。このショートネームは有効なC識別子である必要はありません

[1] AUTOSAR R3.2.5 以前

[2] AUTOSAR R4.0.*

ASCET が生成するコードについては 145 ページの 10.3.2 項「ポートからの受信」を参照してください。

8.6 ポートへのサーバーリクエストへの応答

ASCET は、クライアント／サーバーインターフェースを提供するソフトウェアコンポーネント内に、そのインターフェース内の各オペレーションを行うランナブルエンティティを定義します。これらのランナブルエンティティは、そのソフトウェアコンポーネント上の P ポート用サーバーになります。

RTE がサーバーとして扱うランナブルエンティティは、<OPERATION-INVOKED-EVENT>に結びつけられている必要があります。この RTE イベントにより、RTE は実行時にクライアントの要求に応えて所定のランナブルエンティティを呼び出すことができます。また<OPERATION-INVOKED-EVENT>には、サーバーインターフェースに対するどのオペレーション要求によりランナブルエンティティが起動されるようにするかを指定する必要があります。

次の例では、インターフェース型 CSInterface の P ポート Server 上で MaximumValue というオペレーションが呼び出されたときにランナブル Server_MaximumValue が実行されるように設定されています。以下の説明も参照してください。

1. 6.3 項「クライアント／サーバー」（70 ページ）― クライアントインターフェース CSInterface の作成について
2. 7.1.1 項「P ポート（提供ポート）」（87 ページ）― P ポート Server の作成について
3. 8.1.2 項「オペレーション呼び出しイベント」（103 ページ）― オペレーション呼び出しイベントについての詳しい説明

```
<INTERNAL-BEHAVIOR>
  <SHORT-NAME>bSwc</SHORT-NAME>
  <COMPONENT-REF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE">
  /ASCET_swcomponents/Impl/Swc</COMPONENT-REF>
  <EVENTS>
    ...
    <OPERATION-INVOKED-EVENT>
      <SHORT-NAME>OpInvEvent_Server_MaximumValue</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
      /ASCET_swcomponents/Impl/bSwc/Server_MaximumValue
      </START-ON-EVENT-REF>
      <OPERATION-IREF>
        <P-PORT-PROTOTYPE-REF DEST="P-PORT-PROTOTYPE">
        /ASCET_swcomponents/Impl/Swc/Server</P-PORT-PROTOTYPE-REF>
        <OPERATION-PROTOTYPE-REF DEST="OPERATION-PROTOTYPE">
        /ASCET_interfaces/Impl/CSInterface/MaximumValue
        </OPERATION-PROTOTYPE-REF>
      </OPERATION-IREF>
    </OPERATION-INVOKED-EVENT>
    ...
  </EVENTS>
  <RUNNABLES>
    ...
    <RUNNABLE-ENTITY>
      <SHORT-NAME>Server_MaximumValue</SHORT-NAME>
      <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
      <SYMBOL>Swc_Impl_Server_MaximumValue</SYMBOL>
    </RUNNABLE-ENTITY>
    ...
  </RUNNABLES>
</INTERNAL-BEHAVIOR>
```

コード 77: ARXML コード – サーバーリクエストに応答する内部ビヘイビア（AUTOSAR R3.1.2）

```
<SWC-INTERNAL-BEHAVIOR>
  <SHORT-NAME>bSWC</SHORT-NAME>
  <DATA-TYPE-MAPPING-REFS>
    <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">
        /ASCET_Mappings/DataMappings/Impl</DATA-TYPE-MAPPING-REF>
  </DATA-TYPE-MAPPING-REFS>
  <EVENTS>
    ...
    <OPERATION-INVOKED-EVENT>
      <SHORT-NAME>OpInvEvent_Server_MaximumValue</SHORT-NAME>
      <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
          /ASCET_ComponentTypes/SWC/bSWC/Server_MaximumValue
          </START-ON-EVENT-REF>
      <OPERATION-IREF>
        <CONTEXT-P-PORT-REF DEST="P-PORT-PROTOTYPE">
            /ASCET_ComponentTypes/SWC/Server</CONTEXT-P-PORT-REF>
        <TARGET-PROVIDED-OPERATION-REF DEST="CLIENT-SERVER-OPERATION">
            /ASCET_Interfaces/Impl/CSInterface/MaximumValue
            </TARGET-PROVIDED-OPERATION-REF>
      </OPERATION-IREF>
    </OPERATION-INVOKED-EVENT>
    ...
  </EVENTS>
  <RUNNABLES>
    ...
    <RUNNABLE-ENTITY>
      <SHORT-NAME>Server_MaximumValue</SHORT-NAME>
      <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
          </SW-ADDR-METHOD-REF>
      <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
      <SYMBOL>SWC_Impl_Server_MaximumValue</SYMBOL>
    </RUNNABLE-ENTITY>
    ...
  </RUNNABLES>
</SWC-INTERNAL-BEHAVIOR>
```

コード 78: ARXML コード — サーバーリクエストに応答する内部ビヘイビア（AUTOSAR R4.0.2）

<OPERATION-INVOKED-EVENT>の名前は<SHORT-NAME>エレメントで定義されます。この名前は、このイベントを参照する他のエレメントの中で使用されます。有効な C 識別子である必要はありません。

8.6.1 サーバーの並行呼び出し

サーバーとして機能するランナブルが並行して呼び出されるように記述されている場合、RTE は、最適化機能により、同じ ECU 上のクライアントによる呼び出しを関数の「直接呼び出し」にすることができます。この場合、キューイングは不要（または不可能）で、サーバーに対する複数の呼び出しを並行して発生させることができるようになります。

この際 RTE ジェネレータは、どのランナブルエンティティを並行的に呼び出せるようにするかを知っておく必要があります。

サーバーの並行呼び出しを可能にする:

- ソフトウェアコンポーネントエディタの“Outline”タブで、サーバーランナブル `Server_MaximumValue` を選択します。

このサーバーランナブル Server_MaximumValue は、7.1.1 項「P ポート」で P ポート Server が作成されたときに ASCET により自動挿入されたものです。

“Runnable Signature Editor for: Server_MaximumValue”ウィンドウが開きます。

- “Settings”タブを選択します。
- **Can be Invoked Concurrently** オプションをオンにします。

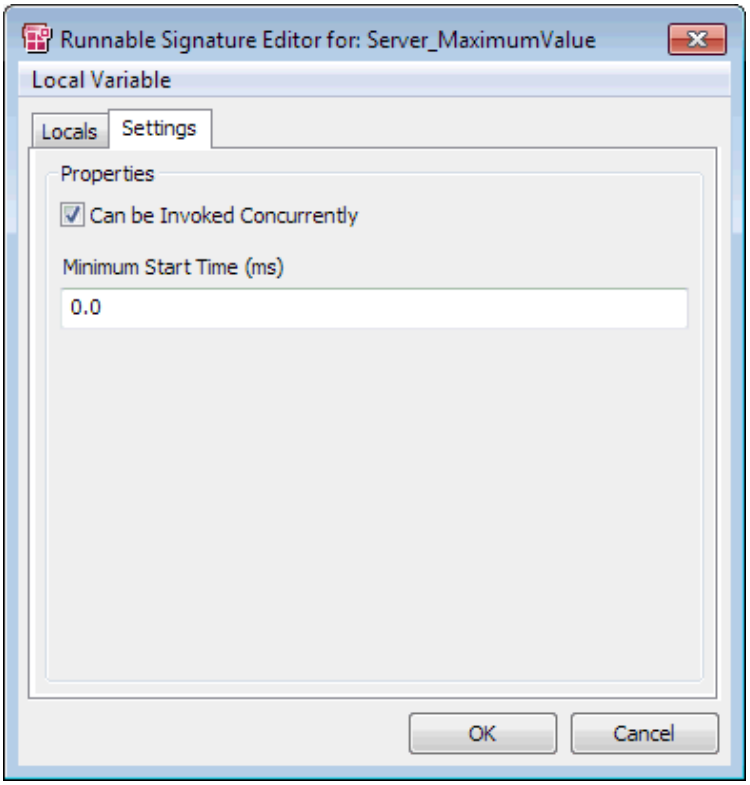


図 51: ランナブル Server_MaximumValue 用の **Can be Invoked Concurrently** オプション

- **OK** をクリックしてランナブルシグネチャエディタを閉じます。

並行呼び出しは、サーバーのランナブルエンティティ内に以下のように定義されます。

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>Server_MaximumValue</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>true</CAN-BE-INVOKED-CONCURRENTLY>
   <SYMBOL>Swc_Impl_Server_MaximumValue</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 79: ARXML コード – 並行呼び出しが可能なサーバーランナブル（AUTOSAR R3.1.2）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>Server_MaximumValue</SHORT-NAME>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>true</CAN-BE-INVOKED-CONCURRENTLY>
   <SYMBOL>SWC_Impl_Server_MaximumValue</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 80: ARXML コード – 並行呼び出しが可能なサーバーランナブル（AUTOSAR R4.0.2）

注記

これらのランナブルは、並行して呼び出されることを前提にして記述されていなければなりません。そうでないと、複数のクライアントがこのサーバーを同時に要求した場合にデータの整合性が保証されなくなります。

8.7 ポートへのクライアントリクエスト

同様に、クライアント／サーバーインターフェースを必要とするソフトウェアコンポーネントには、クライアントとして機能するランナブルエンティティを 1 つ以上定義する必要があります。

ASCET では、複数のクライアントからサーバーに同時にアクセスでき、サーバーがリクエストを処理している間、クライアントはブロックされます。サーバーが処理を終えるとその結果がクライアントに返され、クライアントは処理を再開します。ここでは、クライアントが必ず RTE イベントによりトリガされるようにする必要があります。

ポートにクライアントリクエストを発行する:

- 「クライアントポートを作成する」(95 ページ)の方法で、R ポート `Client` を Swc に追加します。
- R ポート `Client` を"Outline"タブからドラッグし、ソフトウェアコンポーネントエディタの描画エリアにドロップします。
- メソッド `Notification` を非アクティブ化します(96 ページ参照)。
- 符号付き離散変数を挿入して名前を A にし、範囲が`[-32768, 32767]`の `sint16` として実装します。
- A を R ポート `Client` の引数 `InputA` に接続します。
- 符号付き離散変数 B および C を挿入し、A と同じ内容で実装します。
- B を `Client` の引数 `InputB` に接続します。
- C を `Client` の引数 `OutputMaximum` に接続します。
- **RTE Invoke** ボタンを使用し、RTE 呼び出し演算子を作成して描画エリアに配置します。
- オペレーション `MaximumValue` の戻り値を RTE 呼び出し演算子に接続します。
- ツリーペイン内のランナブル `RunnableEntity` を選択し、空のシーケンスコール **InvokeOp** をダブルクリックします。

  ASCET は、**InvokeOp** にランナブルエンティティ `RunnableEntity` のシーケンス番号(ここでは 5)を自動的に割り当てます。

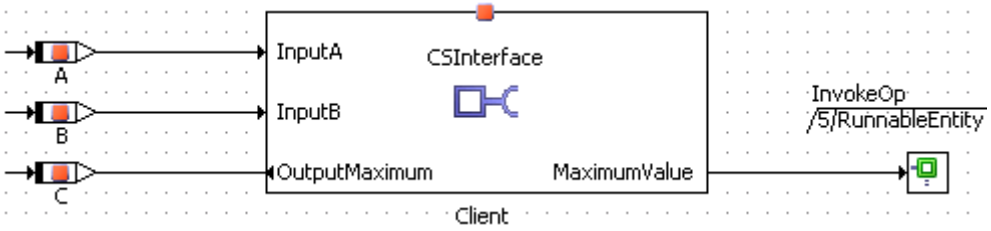


図 52: R ポート `Client` に `MaximumValue(A,B)`を計算して結果を C に格納するよう要求する

複数のランナブルエンティティが同時にサーバーを呼び出す必要がある場合、<SYNCHRONOUS-SERVER-CALL-POINT>エレメントで、クライアントが呼び出すことのできるオペレーションを指定し、呼び出されるすべてのオペレーションについてグローバルの<TIMEOUT>値を定義します。<TIMEOUT>は、サーバーがオペレーション結果を提供するまでのクライアント側の最大待ち時間です。

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   ...
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SERVER-CALL-POINTS>
      <SYNCHRONOUS-SERVER-CALL-POINT>
         <SHORT-NAME>serverCallPoint1</SHORT-NAME>
         <OPERATION-IREFS>
            <OPERATION-IREF>
               <R-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
               /ASCET_swcomponents/Impl/Swc/Client
               </R-PORT-PROTOTYPE-REF>
               <OPERATION-PROTOTYPE-REF DEST="OPERATION-PROTOTYPE">
               /ASCET_interfaces/Impl/CSInterface/MaximumValue
               </OPERATION-PROTOTYPE-REF>
            </OPERATION-IREF>
         </OPERATION-IREFS>
         <TIMEOUT>0</TIMEOUT>
      </SYNCHRONOUS-SERVER-CALL-POINT>
   </SERVER-CALL-POINTS>
   <SYMBOL>RteRunnable_SWC_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 81: ARXML コード – クライアントリクエストが定義されたランナブルエンティティ（AUTOSAR R3.1.2）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   ...
   <SERVER-CALL-POINTS>
      <SYNCHRONOUS-SERVER-CALL-POINT>
         <SHORT-NAME>ServerCallPoint1</SHORT-NAME>
         <OPERATION-IREF>
            <CONTEXT-R-PORT-REF DEST="R-PORT-PROTOTYPE">
                /ASCET_ComponentTypes/SWC/Client</CONTEXT-R-PORT-REF>
            <TARGET-REQUIRED-OPERATION-REF DEST="CLIENT-SERVER-OPERATION">
                /ASCET_Interfaces/Impl/CSInterface/MaximumValue
                </TARGET-REQUIRED-OPERATION-REF>
         </OPERATION-IREF>
         <TIMEOUT>0</TIMEOUT>
      </SYNCHRONOUS-SERVER-CALL-POINT>
   </SERVER-CALL-POINTS>
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 82: ARXML コード – クライアントリクエストが定義されたランナブルエンティティ（AUTOSAR R4.0.2）

<SNCHRONOUS-SERVER-CALL-POINT>には、<SHORT-NAME>エレメントで名前を定義する必要があります。この名前はこの呼び出しポイントを参照する他のエレメントの中で使用されます。これは有効な C 識別子である必要はありませんが、AUTOSAR スキーマが実施する構文チェックをパスするものでなければなりません。

注記

呼び出されるすべてのオペレーションについてのグローバルの<TIMEOUT>値は、ASCET では常に 0 に設定されます。

ASCET が生成するコードについては 153 ページの 10.4.2 項「ポートにクライアントリクエストを発行する」を参照してください。

1 つのランナブルエンティティを、ある 1 つのインターフェースではサーバーとして使用し、別のインターフェースではクライアントとして使用することができます。たとえば、P ポートではソートを行うサーバーリクエストを扱い、R ポートでは補助オペレーションを使用するランナブルエンティティを作成できます。

8.8 インターランナブル変数

非 AUTOSAR プロジェクトでは、ASCET メッセージをプロセス間通信に使用することができます。AUTOSAR ソフトウェアコンポーネントではこれらのメッセージを使用できないため、代わりに「インターランナブル変数」を使用してランナブル間の通信を行います。

インターランナブル変数による通信は、明示的／暗黙的なセンダ／レシーバ通信とセマンティックス的に同等です（62 ページ、6.1 項「センダ／レシーバ」を参照）が、スコープはソフトウェアコンポーネントのインスタンス内に限られます。

インターランナブル変数を定義する:

- ソフトウェアコンポーネントエディタで、**Interrunnable Variable** ボタンを使用してインターランナブル変数を追加します。
  "Properties for Scalar Element: interrunnable"ダイアログボックスが開きます。
- インターランナブル変数の名前を `IRV_explicit` にします。
- "Internal Access" を **Explicit** にします。
- "Basic Type"を選択します（例: `Unsigned Discrete`）。
- **OK** をクリックしてプロパティエディタを閉じます。
- さらに、内部アクセスが **Implicit** のインターランナブル変数 `IRV_implicit` を作成します。
- 両方のインターランナブル変数を `sint8` として実装します(図 **10** 参照)。

AUTOSAR R3.1.5 以前では、インターランナブル変数は<INTER-RUNNABLE-VARIABLE>エレメントで定義されます。<COMMUNICATION-APPROACH> エレメントで、変数が暗黙的アクセスと明示的アクセスのどちらを使用するかが決まります。

```
<INTER-RUNNABLE-VARIABLES>
  <INTER-RUNNABLE-VARIABLE>
    <SHORT-NAME>IRV_explicit</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="INTEGER-TYPE">/AUTOSAR_types/UInt8</TYPE-TREF>
    <COMMUNICATION-APPROACH>EXPLICIT</COMMUNICATION-APPROACH>
    <INIT-VALUE-REF DEST="INTEGER-LITERAL">/ASCET_types/constants/c_uint80/value
    </INIT-VALUE-REF>
  </INTER-RUNNABLE-VARIABLE>
  <INTER-RUNNABLE-VARIABLE>
    <SHORT-NAME>IRV_implicit</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="INTEGER-TYPE">/AUTOSAR_types/UInt8</TYPE-TREF>
    <COMMUNICATION-APPROACH>IMPLICIT</COMMUNICATION-APPROACH>
    <INIT-VALUE-REF DEST="INTEGER-LITERAL">/ASCET_types/constants/c_uint86/value
    </INIT-VALUE-REF>
  </INTER-RUNNABLE-VARIABLE>
</INTER-RUNNABLE-VARIABLES>
```

コード 83: ARXML コード – 明示的インターランナブル変数と暗黙的インターランナブル変数（AUTOSAR R3.1.2）

AUTOSAR R4.0.*では、インターランナブル変数は<VARIABLE-DATA-PROTOTYPE>エレメントで定義されます。明示的インターランナブル変数と暗黙的インターランナブル変数は、<SWC-INTERNAL-BEHAVIOR>内の異なるエレメント、<EXPLICIT-INTER-RUNNABLE-VARIABLES>（コード 84）および <IMPLICIT-INTER-RUNNABLE-VARIABLES>（コード 85）で定義されます。

```
<EXPLICIT-INTER-RUNNABLE-VARIABLES>
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>IRV_explicit</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/RAM
          </SW-ADDR-METHOD-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/UInt8</TYPE-TREF>
    <INIT-VALUE>
      <NUMERICAL-VALUE-SPECIFICATION>
        <VALUE>0</VALUE>
      </NUMERICAL-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
</EXPLICIT-INTER-RUNNABLE-VARIABLES>
```

コード 84: ARXML コード – 明示的インターランナブル変数（AUTOSAR R4.0.2）

```
<IMPLICIT-INTER-RUNNABLE-VARIABLES>
  <VARIABLE-DATA-PROTOTYPE>
    <SHORT-NAME>IRV_implicit</SHORT-NAME>
    <SW-DATA-DEF-PROPS>
      <SW-DATA-DEF-PROPS-VARIANTS>
        <SW-DATA-DEF-PROPS-CONDITIONAL>
          <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/RAM
          </SW-ADDR-METHOD-REF>
          <SW-CALIBRATION-ACCESS>READ-ONLY</SW-CALIBRATION-ACCESS>
          <SW-IMPL-POLICY>STANDARD</SW-IMPL-POLICY>
        </SW-DATA-DEF-PROPS-CONDITIONAL>
      </SW-DATA-DEF-PROPS-VARIANTS>
    </SW-DATA-DEF-PROPS>
    <TYPE-TREF DEST="APPLICATION-PRIMITIVE-DATA-TYPE">
    /ASCET_Types/ApplicationDataTypes/UInt8</TYPE-TREF>
    <INIT-VALUE>
      <NUMERICAL-VALUE-SPECIFICATION>
        <VALUE>6</VALUE>
      </NUMERICAL-VALUE-SPECIFICATION>
    </INIT-VALUE>
  </VARIABLE-DATA-PROTOTYPE>
</IMPLICIT-INTER-RUNNABLE-VARIABLES>
```

コード 85: ARXML コード – 暗黙的インターランナブル変数（AUTOSAR R4.0.2）

各インターランナブル変数の名前は<SHORT-NAME>エレメントで定義する必要があり、これを使用して他のエレメントがインターランナブル変数を参照します。

8.8.1 読取り／書込みアクセス

各ランナブルエンティティについて、そのランナブルエンティティが実行時にインターランナブル変数を読み取るか書き込むかを明示的に定義しておく必要があります。

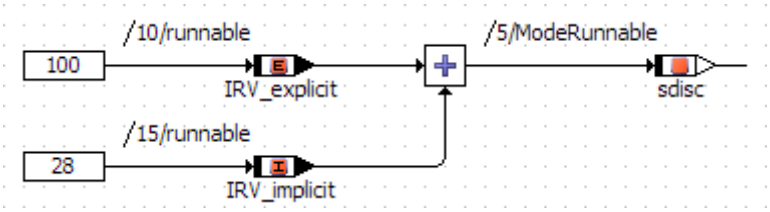


図 53: 2 つのランナブルエンティティにより使用されるインターランナブル変数

AUTOSAR R3.1.5 以前では、アクセスは <READ-VARIABLE-REFS> および<WRITTEN-VARIABLE-REFS>エレメントで宣言されます。図 53 の例は、各ランナブルについての以下のようなディスクリプションにより実現されます。

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>ModeRunnable</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <READ-VARIABLE-REFS>
      <READ-VARIABLE-REF DEST="INTER-RUNNABLE-VARIABLE">
      /ASCET_swcomponents/Impl/bSwc/IRV_explicit</READ-VARIABLE-REF>
      <READ-VARIABLE-REF DEST="INTER-RUNNABLE-VARIABLE">
      /ASCET_swcomponents/Impl/bSwc/IRV_implicit</READ-VARIABLE-REF>
   </READ-VARIABLE-REFS>
   <SYMBOL>Swc_IRV_Impl_ModeRunnable</SYMBOL>
</RUNNABLE-ENTITY>
<RUNNABLE-ENTITY>
   <SHORT-NAME>runnable</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <READ-VARIABLE-REFS>
      <READ-VARIABLE-REF DEST="INTER-RUNNABLE-VARIABLE">
      /ASCET_swcomponents/Impl/bSwc/IRV_implicit</READ-VARIABLE-REF>
   </READ-VARIABLE-REFS>
   <SYMBOL>Swc_Impl_runnable</SYMBOL>
   <WRITTEN-VARIABLE-REFS>
      <WRITTEN-VARIABLE-REF DEST="INTER-RUNNABLE-VARIABLE">
      /ASCET_swcomponents/Impl/bSwc/IRV_explicit</WRITTEN-VARIABLE-REF>
      <WRITTEN-VARIABLE-REF DEST="INTER-RUNNABLE-VARIABLE">
      /ASCET_swcomponents/Impl/bSwc/IRV_implicit</WRITTEN-VARIABLE-REF>
   </WRITTEN-VARIABLE-REFS>
</RUNNABLE-ENTITY>
```

コード 86: ARXML コード – インターランナブル変数への読取り／書込みアクセスを行うランナブルエンティティ（AUTOSAR R3.1.2）

In AUTOSAR R4.0.*では、アクセスは<READ-LOCAL-VARIABLES> および<WRITTEN-LOCAL-VARIABLES>エレメントで宣言されます。図 53 の例は、ランナブル Rrunnable についての以下のようなディスクリプションにより実現されます。

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>runnable</SHORT-NAME>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE</SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <READ-LOCAL-VARIABLES>
      <VARIABLE-ACCESS>
         <SHORT-NAME>Read_IRV_implicit</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
            /ASCET_ComponentTypes/Swc/bSwc/IRV_implicit</LOCAL-VARIABLE-REF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
   </READ-LOCAL-VARIABLES>
   <SYMBOL>Swc_Impl_runnable</SYMBOL>
   <WRITTEN-LOCAL-VARIABLES>
      <VARIABLE-ACCESS>
         <SHORT-NAME>Write_IRV_explicit</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
            /ASCET_ComponentTypes/Swc/bSwc/IRV_explicit</LOCAL-VARIABLE-REF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
      <VARIABLE-ACCESS>
         <SHORT-NAME>Write_IRV_implicit</SHORT-NAME>
         <ACCESSED-VARIABLE>
            <LOCAL-VARIABLE-REF DEST="VARIABLE-DATA-PROTOTYPE">
            /ASCET_ComponentTypes/Swc/bSwc/IRV_implicit</LOCAL-VARIABLE-REF>
         </ACCESSED-VARIABLE>
      </VARIABLE-ACCESS>
   </WRITTEN-LOCAL-VARIABLES>
</RUNNABLE-ENTITY>
```

コード 87: ARXML コード – インターランナブル変数への読取り／書込みアクセスを行うランナブルエンティティ（AUTOSAR R4.0.2）

8.9 排他領域

ソフトウェアコンポーネントは、内部に定義されている複数のランナブルエンティティが共有するデータについて相互排他処理を行う必要がある場合、「排他領域」を使用することができます。

RTE ジェネレータは排他領域設定に従って OS コンフィギュレーションファイルを作成し、排他領域を最適化します。たとえば、ある領域にアクセスするすべてのコンポーネントが同じタスクにマッピングされている場合、その領域全体を排他設定から除外できます。

排他領域は XML コンフィギュレーションに定義され、それらを使用するランナブルエンティティに関連付けられます。

8.9.1 ASCET V6.2 での変更点

これまでのバージョンの ASCET では、SWC 内のメッセージへの読取り／書込みアクセスを保護するために排他的領域が使用され、ASCET_exclusive_area という名前の排他的領域が自動的に作成されていました。

ASCET V6.2 では、SWC 内でメッセージが使用できなくなったため、ASCET モジュールに含まれるメッセージは AUTOSAR エレメントにマッピングする必要があります。そのため、メッセージ保護のための排他的領域も必要なくなり、ASCET_exclusive_area は生成されなくなりました。

8.9.2 設定

排他的領域は、ASCET リソースとして作成します。

排他領域を作成する:

- ソフトウェアコンポーネントエディタで、**Resource** ボタンを使用してリソースを作成し、描画エリアに配置します。
- "Outline" タブでリソースを右クリックし、ショートカットメニューから **Rename** を選択してリソースの名前を SwcExclusiveArea に変更します。

新たに作成されたこの排他領域SwcExclusiveAreaをソフトウェアコンポーネント内で使用するには（8.9.3 項を参照）、そのコンポーネントが使用する排他領域を<INTERNAL-BEHAVIOR>[1] / <SWC-INTERNAL-BEHAVIOR>[2]宣言内の<EXCLUSIVE-AREAS>に指定します。

```
<INTERNAL-BEHAVIOR>
   <SHORT-NAME>bSwc</SHORT-NAME>
   <COMPONENT-REF DEST="APPLICATION-SOFTWARE-COMPONENT-TYPE">
   /ASCET_swcomponents/Impl/Swc</COMPONENT-REF>
   <EVENTS>
      ...
   </EVENTS>
   <EXCLUSIVE-AREAS>
      <EXCLUSIVE-AREA>
         <SHORT-NAME>SwcExclusiveArea</SHORT-NAME>
      </EXCLUSIVE-AREA>
   </EXCLUSIVE-AREAS>
   <RUNNABLES>
      ...
   </RUNNABLES>
</INTERNAL-BEHAVIOR>
```

コード 88: ARXML コード – 排他的領域の定義（AUTOSAR R3.1.2）

```
<SWC-INTERNAL-BEHAVIOR>
   <SHORT-NAME>bSWC</SHORT-NAME>
   <DATA-TYPE-MAPPING-REFS>
      <DATA-TYPE-MAPPING-REF DEST="DATA-TYPE-MAPPING-SET">
          /ASCET_Mappings/DataMappings/Impl</DATA-TYPE-MAPPING-REF>
   </DATA-TYPE-MAPPING-REFS>
   <EXCLUSIVE-AREAS>
      <EXCLUSIVE-AREA>
         <SHORT-NAME>SwcExclusiveArea</SHORT-NAME>
      </EXCLUSIVE-AREA>
   </EXCLUSIVE-AREAS>
   <EVENTS>
      ...
   </EVENTS>
   <RUNNABLES>
      ...
   </RUNNABLES>
</SWC-INTERNAL-BEHAVIOR>
```

コード 89: ARXML コード – 排他的領域の定義（AUTOSAR R4.0.2）

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

つまり、ユーザーが定義する排他領域のスコープはソフトウェアコンポーネントのインスタンスになります。ソフトウェアコンポーネント境界をまたぐ排他領域を定義することはできません。複数のソフトウェアコンポーネントインスタンスに共有され、それらから同時にアクセスされる可能性があるデータは、必ず専用のコンポーネントにカプセル化して、そのデータへのアクセスには通常のセンダ／レシーバ通信またはクライアント／サーバー通信を使用するようにしてください。

内部ビヘイビア内に定義されている各排他領域には、<SHORT-NAME>エレメント内で名前が定義されている必要があります。この名前は、他のエレメント内でこのソフトウェアコンポーネント型を参照し、実行時に排他領域にアクセスするための「ハンドル」を形成するために使用されるもので、有効な C 識別子である必要があります。

さらに、ECU ディスクリプション内の ExclusiveAreaImplementation エレメントにより、排他領域をどのように実装するかを RTE に知らせることができます。

注記

ある排他領域についての ExclusiveAreaImplementation の定義が省略されると、RTE はデフォルトの OS リソース実装ストラテジを使用します。

排他領域と SWC インスタンスの組み合わせごとに、異なる排他領域実装メソッドを設定できます。

注記

InterruptBlocking メソッドを使用している場合、保護されるクリティカルセクションの実行時間が最長になると、最悪の場合にはすべての OS 割込みがブロックされてしまいます。

8.9.3 使用法

<INTERNAL-BEHAVIOR>[1] / <SWC-INTERNAL-BEHAVIOR>[2]セクション内の各ランナブルでは、「排他領域」を使用するかどうか、およびその領域を実行時にどのように使用するかを宣言できます。

ASCET は明示的アクセスとして排他領域を定義します。<RUNNABLE-ENTITY-CAN-ENTER-EXCLUSIVE-AREA>エレメントで、その排他領域へのアクセスに明示的 API が使用されることを指定します。この領域の名前が、生成される API の一部分になります（明示的アクセスは OSEK OS の標準リソースとよく似ています）。

ASCET V6.2 以降において排他的領域は、ユーザー定義された排他的領域内に割り当てられたランナブルエンティティのシーケンスによってのみアクセス可能です。

注記

ASCET V6.2 以降のソフトウェアコンポーネントには、メッセージ、および自動生成される ASCET_exclusive_area は存在しません。

ランナブルのシーケンスを排他領域に割り当てる:

- SwcExclusiveArea のシーケンスコール reserve を編集し、メソッド RunnableEntity のシーケンス番号 8 をセットします。
- SwcExclusiveArea のシーケンスコール release を編集し、メソッド RunnableEntity のシーケンス番号 22 をセットします。

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

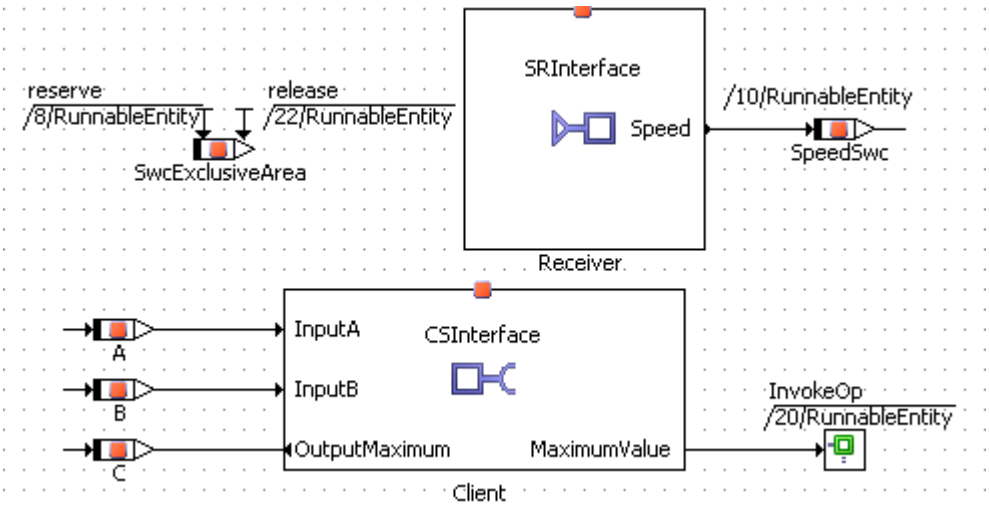


図 54: RunnableEntity 内での排他的領域 SwcExclusiveArea の使用

コード 90 およびコード 91 のように、<RUNNABLE-ENTITY>エレメントの定義内に SwcExclusiveArea への参照が生成されます。

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   <CAN-ENTER-EXCLUSIVE-AREA-REFS>
      <CAN-ENTER-EXCLUSIVE-AREA-REF DEST="EXCLUSIVE-AREA">
      /ASCET_swcomponents/Impl/bSwc/SwcExclusiveArea
      </CAN-ENTER-EXCLUSIVE-AREA-REF>
   </CAN-ENTER-EXCLUSIVE-AREA-REFS>
   ...
   <SYMBOL>RteRunnable_SWC_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 90: ARXML コード – 排他的領域への参照を含むランナブルエンティティ（AUTOSAR R3.1.2）

```
<RUNNABLE-ENTITY>
   <SHORT-NAME>RunnableEntity</SHORT-NAME>
   <CAN-ENTER-EXCLUSIVE-AREA-REFS>
      <CAN-ENTER-EXCLUSIVE-AREA-REF DEST="EXCLUSIVE-AREA">
          /ASCET_ComponentTypes/SWC/bSWC/SwcExclusiveArea
          </CAN-ENTER-EXCLUSIVE-AREA-REF>
   </CAN-ENTER-EXCLUSIVE-AREA-REFS>
   <MINIMUM-START-INTERVAL>0.0</MINIMUM-START-INTERVAL>
   <SW-ADDR-METHOD-REF DEST="SW-ADDR-METHOD">/ASCET_AddrMethods/CODE
       </SW-ADDR-METHOD-REF>
   <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY>
   ...
   <SYMBOL>RteRunnable_Swc_RunnableEntity</SYMBOL>
</RUNNABLE-ENTITY>
```

コード 91: ARXML コード – 排他的領域への参照を含むランナブルエンティティ（AUTOSAR R4.0.2）

ASCET が生成するコードについては、10.7 項「排他領域による並行処理制御」（162 ページ）を参照してください。

9 モード

前章までは AUTOSAR のソフトウェアコンポーネントの定義と設定を行いましたが、本章ではソフトウェアコンポーネントがランナブルエンティティの実行を制御するために使用する「アプリケーションモード」を定義していきます。

本章は、以下の各項で説明されている「モード」について、さらに詳しく説明するものです。

- 6.2 項「モードスイッチ」(67 ページ)
- 8.1.3 項「モード切り替えイベント」(105 ページ)

9.1 モードの定義

「モード」は ASCET_types という AUTOSAR パッケージの<MODE-DECLARATION-GROUP>エレメント内で宣言されます。ASCET_types にはソフトウェアコンポーネント固有の型が入っています。

AUTOSAR R3.1.5 以前では、ASCET_types パッケージはソフトウェアコンポーネントの型ファイル(生成される Types.arxml ファイル)に格納されます。

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_types</SHORT-NAME>
   <DESC></DESC>
   <ELEMENTS>
      <MODE-DECLARATION-GROUP>
         ...
      </MODE-DECLARATION-GROUP>
   </ELEMENTS>
</AR-PACKAGE>
```

コード 92: ARXML コード – モード宣言グループ(AUTOSAR R3.1.2)

AUTOSAR R4.0.*では、ASCET_Types パッケージはソフトウェアコンポーネントのアプリケーション型ファイル(生成される SWC_appltypes.arxml ファイル)に格納されます。

```
<AR-PACKAGE>
   <SHORT-NAME>ASCET_Types</SHORT-NAME>
   <AR-PACKAGES>
      <AR-PACKAGE>
         <SHORT-NAME>ApplicationDataTypes</SHORT-NAME>
         <ELEMENTS>
            <MODE-DECLARATION-GROUP>
               ...
            </MODE-DECLARATION-GROUP>
            ...
         </ELEMENTS>
      </AR-PACKAGE>
   </AR-PACKAGES>
</AR-PACKAGE>
```

コード 93: ARXML コード – モード宣言グループ(AUTOSAR R4.0.2)

<MODE-DECLARATION-GROUP>エレメントで 1 つまたは複数のモードが宣言され、それらのモードがインターフェース宣言に使用されます。

モードグループを作成する:

- コンポーネントマネージャで **Insert → AUTOSAR →Mode Group** を選択します。

- このモードグループの名前を OnOffMode にします。
- 2 つのモード、off および on を作成します。作成方法は 67 ページを参照してください。

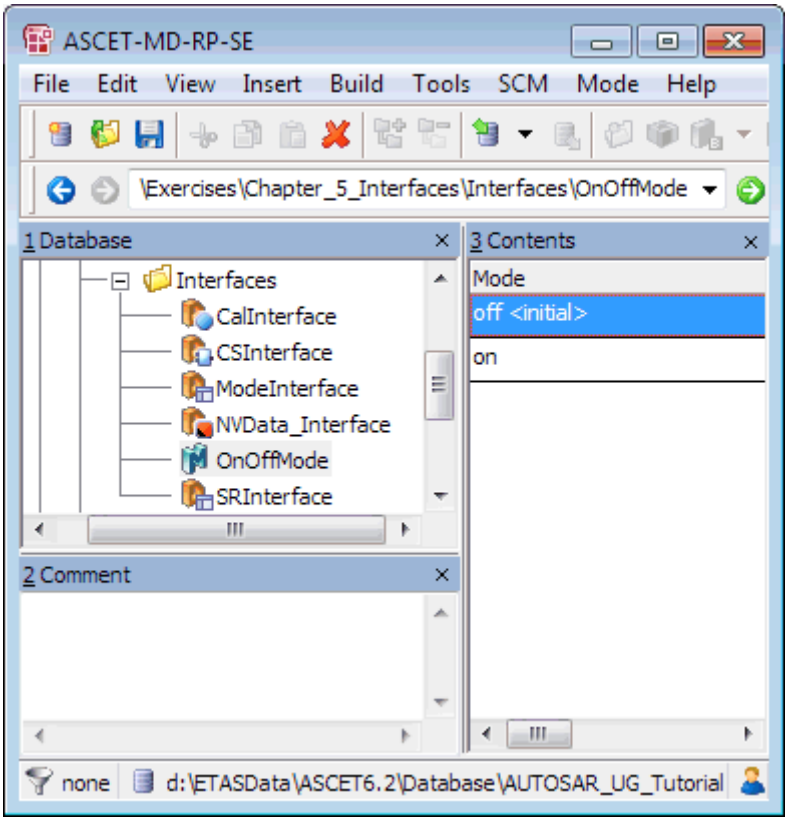


図 55: モード宣言グループ OnOffMode

注記

モード宣言グループは ASCET の列挙型データに似ていますが、列挙型データとは異なり、値を明示的に設定することはできません。

ASCET は AUTOSAR パッケージ ASCET_types 内に<MODE-DECLARATION-GROUP>を宣言します。ARXML コードの例は、68 ページのコード 29（AUTOSAR R3.1.2）または 68 ページのコード 30（AUTOSAR R4.0.2）を参照してください。

<MODE-DECLARATION-GROUP>エレメント内の 1 つのモードが、<INITIAL-MODE-REF>により、そのグループの初期モードとしてマークされます。初期モードの ENTRY に関連付けられているモード切り替えイベントは、Rte_Start によって RTE が起動された際、RTE によりトリガされます。1 つの<MODE-DECLARATION-GROUP>を複数のモードスイッチインターフェースで使用（参照）できるので、複数のソフトウェアコンポーネントからの使用が可能になります。

9.2 モード通信

モードはモードスイッチインターフェースを介して伝えられます（67 ページの 6.2 項「モードスイッチ」を参照）。

ASCET では、モードスイッチインターフェースは、モードグループを含むセンダ／レシーバインターフェースコンポーネントとして実現されます。

AUTOSAR R3.1.5 以前では、各モードスイッチインターフェースには必要に応じて任意の数のモード宣言グループプロトタイプ（インターフェースにより通信される AUTOSAR モード）を定義することができます。

AUTOSAR R4.0.*では、各モードスイッチインターフェースに必ず 1 つのモード宣言グループプロトタイプを定義する必要があります。

モード宣言グループプロトタイプは、特定のモード宣言グループのプロトタイプを定義するものです。

モードグループインターフェースを作成する:

- コンポーネントマネージャで、**Insert → AUTOSAR → SenderReceiver Interface** を選択します。
- このセンダ／レシーバインターフェースの名前を `ModeInterface` にします。
- `ModeInterface` をダブルクリックします。

  “Sender Receiver Interface Editor for: ModeInterface”ウィンドウが開きます。
- **Insert → Component** を選択します。

  “Select Item ...”ダイアログボックスが開きます。
- “Select item...”ダイアログボックスの“1 Database”または“1 Workspace”ペインでモードグループインターフェース `OnOffMode` を選択します（69 ページの図 20 参照）。
- **OK** をクリックして“Select Item...”ダイアログボックスを閉じ、`OnOffMode` を `ModeInterface` に挿入します。

  “Properties for Element: OnOffMode”ダイアログボックスが開きます。
- **OK** をクリックしてデフォルトの名前とコメントを確定します。

  モードグループインターフェース `ModeInterface` は図 21（69 ページ）のようになります。

モードスイッチインターフェース内のモード宣言グループプロトタイプは、70 ページのコード 31（AUTOSAR R3.1.5 以前）または 70 ページのコード 32（AUTOSAR R4.0.*）のような構造で定義されます。AUTOSAR R3.1.5 以前では、各モードグループは <MODE-DECLARATION-GROUP-PROTOTYPE> エレメントで定義され、これらのエレメントがすべて 1 つの<MODE-GROUPS>エレメント内で定義 されている必要があります。

AUTOSAR R4.0.*では、各モードグループは<MODE-GROUP>エレメントで定義されます。

各<MODE-DECLARATION-GROUP-PROTOTYPE>/<MODE-GROUP>には以下のものが定義されている必要があります。

- <SHORT-NAME> - アイテム参照用の名前
- <TYPE-TREF> - モード宣言グループへの参照

AUTOSAR R3.1.5 以前では、1 つのセンダ／レシーバインターフェースコンポーネントで<DATA-ELEMENTS> および<MODE-GROUPS> の両方を同じように宣言することができますが、実際には、1 つのインターフェースにはデータエレメントまたはモードグループのいずれかのみを使用することをお勧めします。

AUTOSAR R4.0.*では、1 つのセンダ／レシーバインターフェースコンポーネントにはデータエレメントとデータエレメントのいずれか一方しか含めることができません。両方のエレメントを混在させると、コード生成エラーが発生します。

9.3 モードの使用

あるソフトウェアコンポーネントを、所定のモード切り替えによって起動されるようにすることができます。本項では、ソフトウェアコンポーネント内でのモードの使用法について説明します。

ソフトウェアコンポーネント内にモードグループインターフェースを挿入する:

- 3.1.2 項(22 ページ)に説明されている方法で、プロジェクトを作成してセットアップします。
- 26 ページ「AUTOSAR ソフトウェアコンポーネントをプロジェクトに挿入する」を参考に、ソフトウェアコンポーネント Swc をプロジェクトに挿入します。
- プロジェクトエディタウィンドウの“Outline”タブで、Swc をダブルクリックしてソフトウェアコンポーネントエディタを開きます。
- ソフトウェアコンポーネントエディタで **Insert → Component** を選択します。

  “Select item...”ダイアログボックスが開きます。
- “Select item...”ダイアログボックスの“1 Database”または“1 Workspace”ペインでインターフェース `ModeInterface` を選択して **OK** をクリックします。

  “Properties for complex element: ModeInterface”ダイアログボックスが開きます。
- **OK** をクリックしてデフォルト設定をそのまま確定します。

9.3.1 ソフトウェアコンポーネントの初期化と終了

AUTOSAR のモードを使用すると、RTE 起動時に内部データ構造の初期化などを行う処理を実行することできます。同様に、システム終了時にはデータを保存したりオペレーション結果をログに記録したりするような処理を実行することも可能です。

各モード宣言グループには「初期モード」を定義し、システム開始時に、<MODE-SWITCH-EVENT>[1] / <SWC-MODE-SWITCH-EVENT>[2]で初期化モードの入り口として定義されたランナブルが起動されるようにします。

モード切り替えイベントを作成する:

- “Software Component Editor for: Swc”ウィンドウの“Event Specification”タブを選択します。
- **Event →Add Event** を選択し、追加されたイベントの名前を `ModeEvent` にします。
- "Event Kind"コンボボックスから `ModeSwitch` を選択します。
- モード切り替えについて以下のように設定します(105 ページの図 42 を参照)。
  - Activation: `entry`
  - Assigned Mode: `On::OnOffMode`

初期モード入口用の<MODE-SWITCH-EVENT> / <SWC-MODE-SWITCH-EVENT>を宣言することにより、ソフトウェアコンポーネント内のランナブルエンティティが RTE 開始時に起動されるようにすることができます。

9.3.2 モード切り替え時にランナブルエンティティをトリガする

設定されているモード切り替えイベントを、他のすべてのイベントと同様にソフトウェアコンポーネントの<INTERNAL-BEHAVIOR>[1] / <SWC-INTERNAL-BEHAVIOR>[2] エレメント内で使用して、ランナブルエンティティをモードの入口か出口のどちらかに起動されるようにすることができます。

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

ランナブルエンティティを作成する:

- “Software Component Editor for: Swc”ウィンドウの“Outline”タブで、ダイアグラム（例: Main）を選択します。
- **Insert → Runnable** を選択し、そのランナブルの名前を ModeRunnable にします。

ランナブルエンティティの詳細については、8.2 項「ランナブルエンティティ」を参照してください。

RunnableEntity を入口用に定義する場合、このランナブルエンティティはカテゴリ 1 である必要があります。つまり、RTE のブロッキング呼び出しを行ったり、他のアプリケーションコンポーネントにアクセスしたりすることはできません。

同様に、出口用に定義されたシステムの場合、ソフトウェアコンポーネントには、データを格納して終了（ターミネーション）をログに記録するなどの処理が必要になります。「終了」は、「シャットダウン」に関連付けられた新しいモードへのトランジションである、という見方をすれば、基本原理は初期化の場合と同じです。

ランナブルにモード切り替えイベントを割り当てる:

- “Software Component Editor for: Swc”ウィンドウの“Event Specification”タブを選択します。
- “Events”フィールドから ModeEvent を選択します。
- “Runnables”フィールドからランナブル ModeRunnable を選択します。
- **Event →Assign Event** を選択するか、または**>>**ボタンを押します。

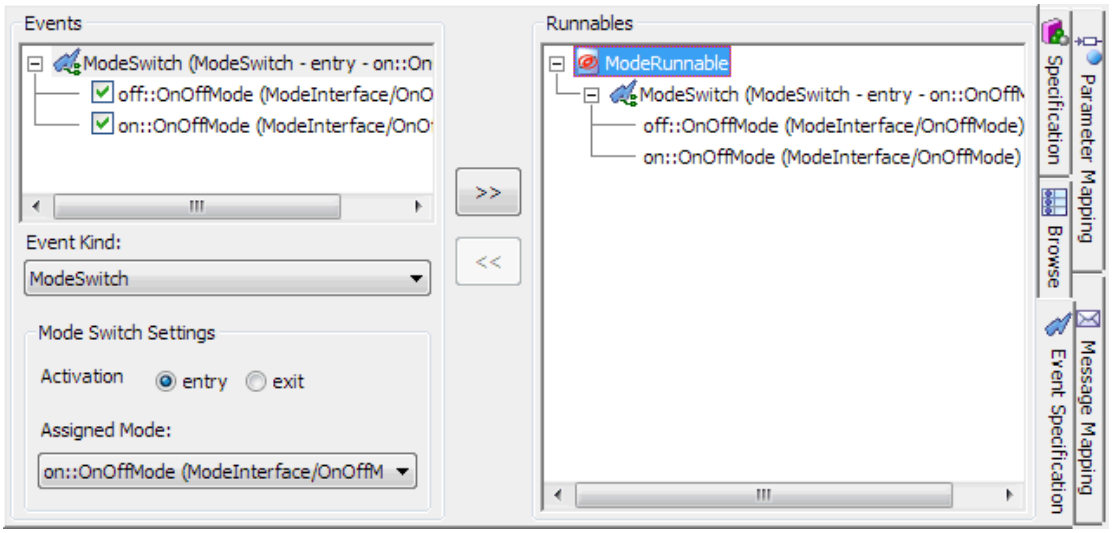


図 56: ModeRunnable に ModeEvent を割り当てる

モード切り替えイベントがランナブルエンティティにマッピングされると、ASCETはコンフィギュレーション言語で<MODE-SWITCH-EVENT>[1] / <SWC-MODE-SWITCH-EVENT>[2]エレメントを生成します。106 ページのコード 63（AUTOSAR R3.1.2）または 106 ページのコード 64（AUTOSAR R4.0.2）を参照してください。

各<MODE-SWITCH-EVENT>/<SWC-MODE-SWITCH-EVENT>エレメントには以下の 3 つの情報が定義されます。

1. <START-ON-EVENT-REF>エレメントで、起動すべきランナブルエンティティが定義されます。これは同じソフトウェアコンポーネント型内のランナブルエンティティへの参照です。

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

2. <ACTIVATION>エレメントでは、ランナブルエンティティがモードの入口と出口のどちらでトリガされるかが定義されます。ASCET は ENTRY または EXIT というテキストをサポートしています。モード切り替えイベントはモードの入口か出口のどちらにも適用できますが、1 つのモード切り替えイベントを入口と出口の両方に適用することはできません。ランナブルの起動が入口と出口の両方に必要な場合は、モード切り替えイベントを 2 つ定義する必要があります。
3. <MODE-IREF>エレメントで、モード切り替えイベントに関連付けられるモードが定義されます。<MODE-IREF>には 3 つの参照(ポートプロトタイプ、モード宣言グループプロトタイプ、そのモード宣言グループプロトタイプを型分けするモード宣言グループ)が含まれている必要があります。

<MODE-DECLARATION-GROUP>エレメント内の 1 つのモードが、そのグループの初期モードとしてマークされます。Rte_Start により RTE が起動されると、グループ内で初期モードの入口に関連付けられているすべてのモード切り替えイベントがトリガされます。

注記

同じモードの入口(または出口)で 2 つ以上のランナブルエンティティがトリガされる場合、各ランナブルエンティティの実行順序は指定できません。移植性を持たせるためには、システムがそれらの実行順序に影響されないようにしてください。

9.3.3 モードの無効化

<MODE-DEPENDENCY>[1] / <DISABLED-MODE-IREFS>[2]エレメントにより、イベントのビヘイビアをモードごとに変えることができます。これにより、アクティブモードに応じてランナブルエンティティの起動を抑止することができます。

ランナブルの起動を無効にする:

- "Software Component Editor for: Swc"ウィンドウの"Event Specification"タブを選択します。
- "Events"フィールドからイベント ModeEvent を選択します。
- モード off を無効にします。

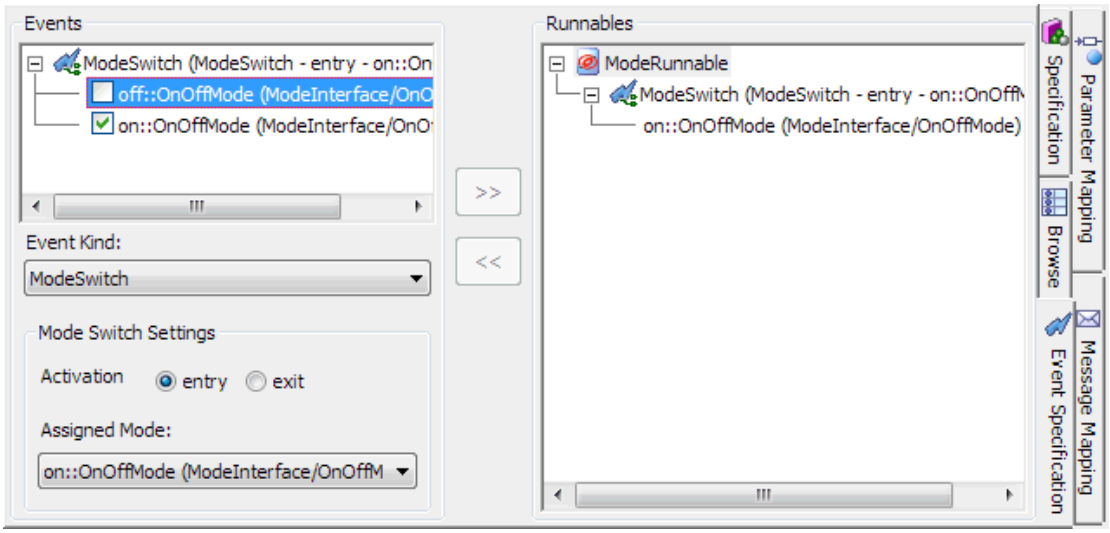


図 57: ModeEvent のモード off を無効にする

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

以下の<MODE-DEPENDENCY>[1] / <DISABLED-MODE-IREFS>[2]エレメントに、この無効化されたモードが記述されています。

```
<MODE-SWITCH-EVENT>
   <SHORT-NAME>ModeEvent</SHORT-NAME>
   <MODE-DEPENDENCY>
      <DEPENDENT-ON-MODE-IREFS>
         <DEPENDENT-ON-MODE-IREF>
            <R-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
            /ASCET_swcomponents/Impl/Swc/ModeInterface</R-PORT-PROTOTYPE-REF>
            <MODE-DECLARATION-GROUP-PROTOTYPE-REF
            DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
            /ASCET_interfaces/Impl/ModeInterface/OnOffMode
            </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
            <MODE-DECLARATION-REF DEST="MODE-DECLARATION">/ASCET_types/OnOffMode/off
            </MODE-DECLARATION-REF>
         </DEPENDENT-ON-MODE-IREF>
      <DEPENDENT-ON-MODE-IREFS>
   </MODE-DEPENDENCY>
   <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
   /ASCET_swcomponents/Impl/bSwc/ModeRunnable</START-ON-EVENT-REF>
   <ACTIVATION>ENTRY</ACTIVATION>
   <MODE-IREF>
      <R-PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">
      /ASCET_swcomponents/Impl/Swc/ModeInterface</R-PORT-PROTOTYPE-REF>
      <MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST="MODE-DECLARATION-GROUP-PROTOTYPE">
      /ASCET_interfaces/Impl/ModeInterface/OnOffMode
      </MODE-DECLARATION-GROUP-PROTOTYPE-REF>
      <MODE-DECLARATION-REF DEST="MODE-DECLARATION">/ASCET_types/OnOffMode/on
      </MODE-DECLARATION-REF>
   </MODE-IREF>
</MODE-SWITCH-EVENT>
```

コード 94: ARXML コード – 無効化されたモードを含むモード切り替えイベントの定義（AUTOSAR R3.1.2）

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

```
<SWC-MODE-SWITCH-EVENT>
   <SHORT-NAME>ModeEvent</SHORT-NAME>
   <DISABLED-MODE-IREFS>
      <DISABLED-MODE-IREF>
         <CONTEXT-PORT-REF DEST="R-PORT-PROTOTYPE">
         /ASCET_ComponentTypes/SWC/ModeInterface</CONTEXT-PORT-REF>
         <CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST=
         "MODE-DECLARATION-GROUP-PROTOTYPE">
         /ASCET_Interfaces/Impl/ModeInterface/OnOffMode
         </CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF>
         <TARGET-MODE-DECLARATION-REF DEST="MODE-DECLARATION">
         /ASCET_Types/ApplicationDataTypes/OnOffMode/off
         </TARGET-MODE-DECLARATION-REF>
      </DISABLED-MODE-IREF>
   </DISABLED-MODE-IREFS>
   <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">
   /ASCET_ComponentTypes/SWC/bSWC/ModeRunnable</START-ON-EVENT-REF>
   <ACTIVATION>ON-ENTRY</ACTIVATION>
   <MODE-IREFS>
      <MODE-IREF>
         <CONTEXT-PORT-REF DEST="R-PORT-PROTOTYPE">
         /ASCET_ComponentTypes/SWC/ModeInterface</CONTEXT-PORT-REF>
         <CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF DEST=
         "MODE-DECLARATION-GROUP-PROTOTYPE">
         /ASCET_Interfaces/Impl/ModeInterface/OnOffMode
         </CONTEXT-MODE-DECLARATION-GROUP-PROTOTYPE-REF>
         <TARGET-MODE-DECLARATION-REF DEST="MODE-DECLARATION">
         /ASCET_Types/ApplicationDataTypes/OnOffMode/on
         </TARGET-MODE-DECLARATION-REF>
      </MODE-IREF>
   </MODE-IREFS>
</SWC-MODE-SWITCH-EVENT>
```

コード 95: ARXML コード – 無効化されたモードを含むモード切り替えイベントの定義（AUTOSAR R4.0.2）

<MODE-DEPENDENCY>[1] / <DISABLED-MODE-IREFS>[2]エレメント内に指定されているモードがアクティブになっても、RTEはランナブルを起動しません。つまり起動要求は破棄されます。

モードインスタンスの詳しい実装方法については、『RTA-RTE User Guide』を参照してください。

[1] AUTOSAR R3.1.5 以前

[2] AUTOSAR R4.0.*

10 ソフトウェアコンポーネントの実装

本章では、ASCET でソフトウェアコンポーネントをモデリングして、RTE が必要とするオブジェクトを宣言する方法、および RTE ジェネレータにより生成される RTE API の使用法について説明します。

10.1 基本概念

10.1.1 ネームスペース

グローバルネームスペースで見ることのできるすべての RTE シンボル（関数名やグローバル変数など）には、Rte_ または RTE_ というプレフィックスが付加されます

注記

ネームスペースの衝突を防ぐため、ユーザーは Rte_または RTE_というプレフィックスを使用するシンボルは作成しないでください。

10.1.2 ランナブルの命名規則

RTE ジェネレータは、ユーザーのランナブルエンティティを起動するコードを生成します。この際、RTE の内部メカニズムは、定義済みインターフェースを通じてユーザーのコードにアクセスできなければなりません。

ユーザーのランナブルエンティティ内に定義された名前を持つ各ランナブルエンティティについて、<SYMBOL>で定義する必要があります。すべてのランナブルエンティティが定義されていないと、コンパイルにおいてユーザーアプリケーションをリンクして ECU の実行イメージを構成する際、エラーが発生し、未定義のランナブルエンティティのエントリポイントがエラーメッセージとして報告されます。

ランナブルエンティティは、要求された時点で、RTE が生成したコードによって実行されます。ランナブルエンティティ用のエントリポイントを提供する関数をアプリケーションソフトウェアコンポーネントから直接呼び出すことはできません。

10.1.3 APIの命名規則

RTE の API コールの名前は、各ソフトウェアコンポーネント用に、RTE ジェネレータの入力に由来する名前を使用して生成されます。RTE API は各ソフトウェアコンポーネント用に一貫したインターフェースを提供し、RTE ジェネレータがさまざまな実装条件において API 機能を提供できるようにします。

各 API コール名は以下のように構成されます。

- プレフィックス Rte_
- 機能（read、write など）
- 以下のいずれか
  - ポート名とデータアイテム名（センダ／レシーバ）またはこの呼び出しの処理を決定するオペレーション名（クライアント／サーバー）
  - この呼び出しの処理対象となるオブジェクト（排他領域など）の名前

従って、ポートによる通信を伴う RTE の API コールの名前は以下の形式になります。

```
Rte_StatusType Rte_<API call name>_<port>_<dataitem/operation>
```

それ以外の RTE API の名前は以下の形式になります。

```
Rte_StatusType Rte_<API call name>_<object name>
```

10.1.4 APIの引数受渡しのメカニズム

RTE の API コールには 1 つまたは複数の引数を指定できます。API 引数（存在する場合）は、以下の 3 種類に分類されます。

- 入力（"in"）引数 – AUTOSAR 基本データ型（ただし文字列型以外）のすべての入力引数は「値渡し」で、文字列型および他の複合データ型の入力引数（つまり、レコードまたは配列）は参照渡しです。

  AUTOSAR では文字列型は基本データ型として規定されていますが、文字列型はサイズが不定なので値渡しが行えないため、複合データ型と同様に扱われます。

  入力引数は読み取り専用です。
- 出力（"out"）引数 – 出力引数は、API 関数で引数値を変更できるように、常に「参照渡し」です。

  出力引数は書き込み専用です。
- 入出力（"in/out"）引数 – 入出力引数の場合は「参照渡し」ですが、「非同期クライアント／サーバー呼び出し」の場合は例外で、基本データ型（文字列型以外）は `Rte_call` には値渡し、`Rte_Result` には参照渡しとなります。

  入出力引数は、呼び出される API 関数から読み取りも書き込みも行えます。

注記

ASCET は、プロジェクトプロパティで指定された XML コンフィギュレーションファイル内に API 引数の識別子を設定します（24 ページの「メモリセクション定義ファイルを定義する」を参照）。なお、AUTOSAR メモリセクションの標準的なコンフィギュレーションが、`memorySections_Autosar.xml` および `memorySections_Autosar4.xml` というサンプルファイルに格納されています。

AUTOSAR プロジェクトのコードを生成する際、ASCET は指定された XML コンフィギュレーションファイルに定義されているメモリセクションをロードします。XML コンフィギュレーションファイルの内容を変更した場合、コード生成開始前にユーザーが **Build →Touch →Recursive** を実行した場合に限りその変更内容がコード生成に反映されます。

10.2 アプリケーションソースコード

ASCET は C コードジェネレータですが、RTE も C コードを生成します。現時点では、ASCET V6.1 はシングルインスタンスのソフトウェアコンポーネントをサポートしています。

10.2.1 アプリケーションヘッダファイル

ASCET で生成された各ソフトウェアコンポーネントには、RTE 設定時に作成されたアプリケーションヘッダファイルがインクルードされます。

```
#include "Rte_SoftwareComponentName.h"

/* Component implementation for "SoftwareComponentName" */
```

コード 96: C コード – アプリケーションヘッダファイルをインクルードする

RTE API はソフトウェアコンポーネント型ごとに固有なので、コンポーネントの全体または一部が定義された各ソースコードファイル用のプリケーションヘッダファイルにのみインクルードされます。lly).

注記

ASCET は、生成したコードを保存用ディレクトリにエクスポートするときにヘッダファイルをアプリケーションソフトウェアにインクルードします（3.1.4 項「コード生成」の、プロジェクトのコードを生成する方法を参照）。ユーザーは、コード生成ディレクトリ内に保存された中間ファイルは使用しないでください。

アプリケーションヘッダファイルに含まれている「API マッピング」はソフトウェアコンポーネントごとに異なる可能性があるので、1 つのソースモジュールに複数のアプリケーションヘッダファイルをインクルードすることはできません。RTE ジェネレータにより生成されるヘッダファイルでは、このような複数ファイルのインクルードに対する保護策が講じられています。

コンポーネント型固有のヘッダファイルにはそのコンポーネントの RTE API が定義されます。

10.2.2 ランナブルエンティティ用のエントリポイントシグネチャ

ユーザーは ASCET で、ソフトウェアコンポーネント内のランナブルのインプリメンテーションをモデリングし、ASCET は、ソフトウェアコンポーネントを実行時に機能させるために必要なすべてのランナブルエンティティのソースコードを生成します。

ASCET は、コンポーネントディスクリプション内に宣言された各<RUNNABLE-ENTITY>用に、エントリポイント（つまり C 関数）を生成します。

```
/* begin region Runnable_Definition_RunnableEntity_ */
/*****************************************************************************
 * BEGIN: DEFINITION OF RUNNABLE 'Swc_Impl_RunnableEntity'
 * ---------------------------------------------------------------------------
 * model name:.................................'RunnableEntity'
 * memory class:...............................'CODE'
 * ---------------------------------------------------------------------------*/
/* messages used by this runnable */
/* public RunnableEntity ()  */

FUNC(void, CODE) Swc_Impl_RunnableEntity (void)
{
   ...
}
```

コード 97: C コード – ランナブルエンティティのエントリポイント

ランナブルエンティティのエントリポイント関数のシグネチャは、以下の実装規則に従っています。

- ユーザー定義の引数はありません。
- 戻り値はありません（つまり void という戻り値型を指定する必要があります）。
- メモリクラスは必ず CODE を使用します。

ランナブルエンティティを実際にトリガするイベントがどのイベントであろうと、「オペレーション呼び出しイベント」以外のすべての RTE イベントは、ランナブルエンティティのエントリポイントとして同じ基本シグネチャを使用します。

ランナブルエンティティがいずれかの<OPERATION-INVOKED-EVENT>に使用されている場合は、引数の追加が必要になる可能性があります。

```
/* begin region Runnable_Definition_Server_MaximumValue_ */
/*****************************************************************************
 * BEGIN: DEFINITION OF RUNNABLE 'Swc_Impl_Server_MaximumValue'
 * ---------------------------------------------------------------------------
 * model name:.....................................'Server_MaximumValue'
 * memory class:...................................'CODE'
 * ---------------------------------------------------------------------------*/
/* messages used by this runnable */
/* public Server_MaximumValue
          (InputA::sdisc;InputB::sdisc;OutputMaximum::sdisc)  */

FUNC(void, CODE) Swc_Impl_Server_MaximumValue (
   /* IN    */ VAR(SInt16, AUTOMATIC)                        InputA,
   /* IN    */ VAR(SInt16, AUTOMATIC)                        InputB,
   /* OUT   */ CONSTP2VAR(SInt16, AUTOMATIC, RTE_APPL_DATA) OutputMaximum
   )
{
   /* Server_MaximumValue: sequence call #5  */
   /* assignment to OutputMaximum: min=-32768, max=32767, hex=phys,
   limit=(maxBitLength: true, assign: true), zero incl.=true  */
   (*OutputMaximum) = ((InputA >= InputB) ? InputA : InputB);
}
```

コード 98: C コード – サーバーランナブルエンティティ

オペレーション呼び出しイベントの結果として呼び出されるランナブルエンティティのエントリポイント関数のシグネチャは、以下の実装規則に従っています。

- サーバーがアプリケーションエラーを規定している場合は、戻り値を持ちます。その場合、`Std_ReturnType` が使用されます。
- 仮引数はオペレーションの各種引数(入力、入出力、出力)です。これらの引数の受渡しは、型に応じて、「値渡し」または「参照渡し」で行われます。
- メモリクラスは必ず `CODE` を使用します。

10.3 センダ／レシーバ通信

非キューイング型センダ／レシーバ通信を処理するための RTE の API コールは、データアクセスのタイプにより異なります

- 明示的アクセスの非キューイング型通信
  - `Rte_Write` による送信
  - `Rte_Read` による受信
  - `Rte_Dread` による受信(AUTOSAR R4.0.*)

  明示的アクセスによる非キューイング型通信は、必要に応じてステータス付きで実装できます。
- 暗黙的アクセスの非キューイング型通信
  - `Rte_IWrite` による送信
  - `Rte_IRead` による受信

  暗黙的 API の場合、ランナブルエンティティを呼び出す際はデータのコピーをローカルにキャッシュしたものを使用し、データの一貫性を守ります。データは、ランナブルエンティティの実行開始前にグローバルキャッシュに読み込まれ、ランナブルエンティティの終了後にグローバルキャッシュから書き出されます。ランナブルエンティティの実行中に書き出し処理が何回行われても、実際のデータ書き出しは 1 回しか行われません。

  RTE は、キャッシュされているデータがランナブルエンティティ実行中に変更されないことを保証します。

暗黙的 API を使用する条件としては、あるランナブルエンティティが呼び出されて所定のデータにアクセスする際、そのランナブルエンティティが 1 回呼び出されて実行される間にそのデータを何回アクセスしても毎回必ず同じ値が読み出されるようにする必要がある場合です。

以下の項では、アプリケーションでこのような通信を使用する方法について説明します。

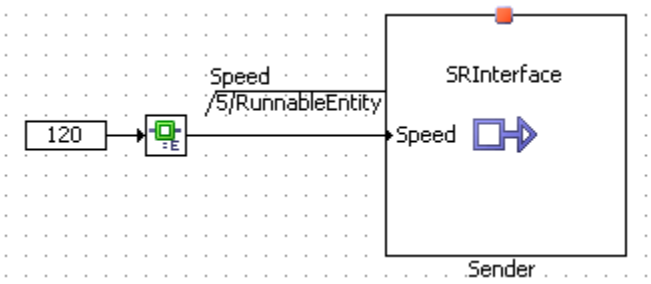
10.3.1 ポートへの送信

明示的通信によるポートへの送信

コンポーネントは他のコンポーネントへのデータ送信に `Rte_Write` コールを使用します。このコールはポートごとに定義され、各コンポーネント用のデータアイテムを受け渡すためのインターフェースとなるので、シグネチャは以下のようになります。

```
Rte_StatusType Rte_Write_<Port>_<DataItem>(DataItemType Data)
```

以下に、8.4.1 項「明示的な送信」に示された例（下図）に対して ASCET が生成する C コードを示します。




```
FUNC(void, CODE) RteRunnable_SWC_RunnableEntity (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   /* RunnableEntity: sequence call #5  */
   _ASCET_RteStatus = Rte_Write_Sender_Speed(120);
}
```

コード 99: C コード – 明示的な送信（8.4.1 項「明示的な送信」の例）

ステータス付き明示的通信によるポートへの送信

明示的アクセスは、必要に応じてステータス付きで実装することもできます。

ステータス付き明示的通信を設定する:

- 8.4.1 項の「明示的な送信」（111 ページ）の例から、プロジェクト `ARProject` のソフトウェアコンポーネント `Swc` を開きます。
- ソフトウェアコンポーネントエディタの描画領域で RTE アクセス演算子を右クリックして、ショートカットメニューから **Access → Explicit with Status** を選択します（図 58）。

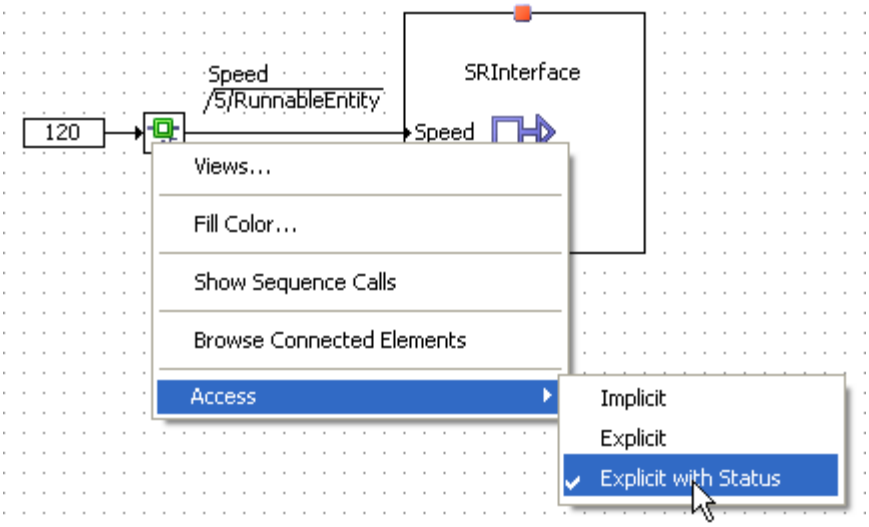


図 58: ステータス付きの明示的通信の設定

- **RTE Status** ボタンを使用し、RTE Status 演算子を作成して描画エリアに配置します。
- いずれかの *** Literal** ボタンを使用し、リテラルを作成して描画エリアに配置します。
- リテラルを編集し（詳細はオンラインヘルプを参照）、39 ページの「Std_ReturnType」の中のいずれかのステータス／エラー値を入力します。

  この例では RTE_E_NO_DATA を使用します。
- 論理変数（例: noData）を追加します。
- 変数のシーケンスコールをコネクタに変換します（詳細はオンラインヘルプを参照）。
- **Equal** 演算子を追加します。
- 各アイテム（リテラル、RTE ステータスブロック、演算子、変数）図 59 のように接続します。
- RTE ステータスブロックの下のピンを変数 noError に接続します。

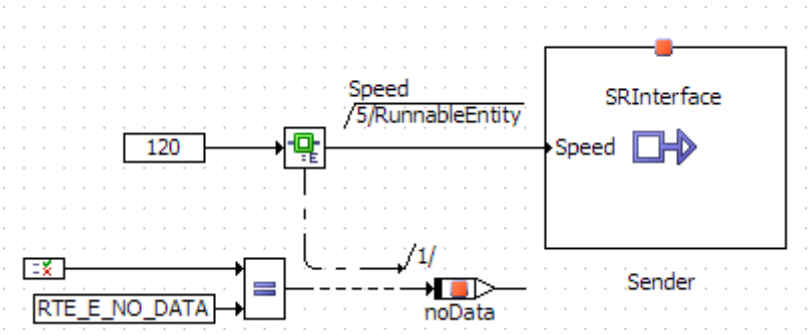


図 59: 値 120 をステータス付きの明示的通信によりセンダポートに送る

この例では、ASCET は以下のような C コードを生成します。

```
#include "Rte_Swc.h"

#define _status Swc_RTE_APPL_DATA.status

FUNC(void, RTE_APPL_CODE) RteRunnable_Swc_RunnableEntity (void)
{
    /* user defined local variables */
    VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

    /* RunnableEntity: sequence call #5 */
    if ((_ASCET_RteStatus = Rte_Write_Sender_Speed(120)) != RTE_E_OK)
    {
        /* RTE_ExplicitWithStatus-block: sequence call #Explicit write error/ Status #1 */
        _status = _ASCET_RteStatus;
    } /* end if */
}
```

コード 100: C コード – ステータス付きの明示的送信

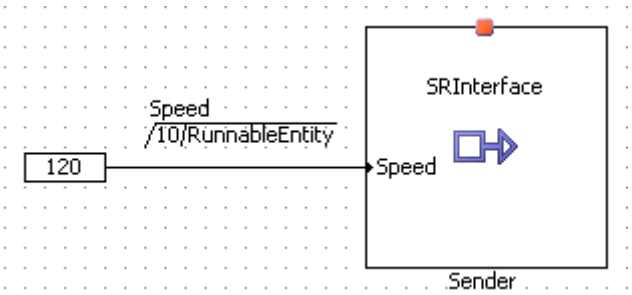
暗黙的通信によるポートへの送信

暗黙的 API の場合、データにアクセスするものとして宣言されているランナブルエンティティへの参照が API 名に含まれています。ランナブルエンティティを作成する際は、正しい API を呼び出すようにするために注意が必要です。Rte_IWrite API は以下のようにデータを読み取ります。

```
Rte_StatusType Rte_IWrite_<runnable>_<port>_<data>( DataItemType Data)
```

キャッシュはランナブルエンティティの起動前に更新されます。Rte_IWrite はキャッシュされたコピーにデータを書き込むので、データの書き込み処理が何回行われても、書き込み後のデータはランナブルエンティティの終了後でないと見ることができません。

8.4.2 項「暗黙的な送信」に示された例（下図）に対して、ASCET は以下のような C コードを生成します。




```
#include "Rte_Swc.h"

FUNC(void, RTE_APPL_CODE) RteRunnable_Swc_RunnableEntity (void)
{
    /* RunnableEntity: sequence call #10 */
   Rte_IWrite_RunnableEntity_Sender_Speed(120);
}
```

コード 101: C コード – 暗黙的送信（8.4.2 項「暗黙的な送信」の例）

10.3.2 ポートからの受信

明示的通信によるポートからの受信

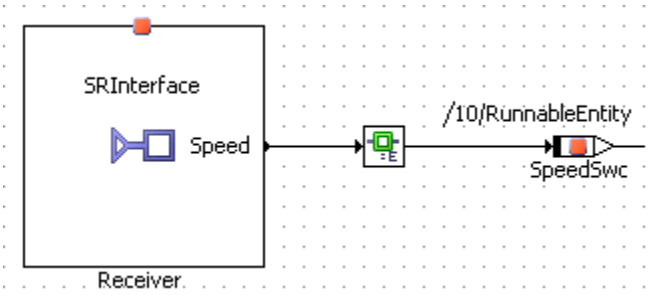
コンポーネントは他のコンポーネントから送られるデータアイテムの受信に Rte_Read コールを使用します。このコールはポートごとに定義され、各コンポーネント用のデータアイテムを受け渡すためのインターフェースとなるため、シグネチャは以下のようになります。

- AUTOSAR R3.1.5 以前

  `Rte_StatusType Rte_Read_<Port>_<DataItem>(DataItemType* Data)`
- AUTOSAR R4.0.*

  `Rte_StatusType Rte_DRead_<Port>_<DataItem>()`

8.5.1 項「明示的な受信」に示された例（下図）に対して、ASCET は以下のような C コードを生成します。




```
...
#define _SpeedSwc Swc_RAM.SpeedSwc
...
FUNC(void, CODE) RteRunnable_SWC_RunnableEntity (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;
   VAR(SInt16, AUTOMATIC) _tSpeed;

   ...
   /* RunnableEntity: sequence call #10  */
   _ASCET_RteStatus = Rte_Read_Receiver_Speed(&(_tSpeed));
   /* assignment to SpeedSwc: min=-32768, max=32767, hex=1phys+0,
   limit=(maxBitLength: true, assign: true), zero incl.=true  */
   _SpeedSwc = _tSpeed;
}
```

コード 102: C コード – 暗黙的受信（8.5.1 項「明示的な受信」– AUTOSAR R3.1.2）

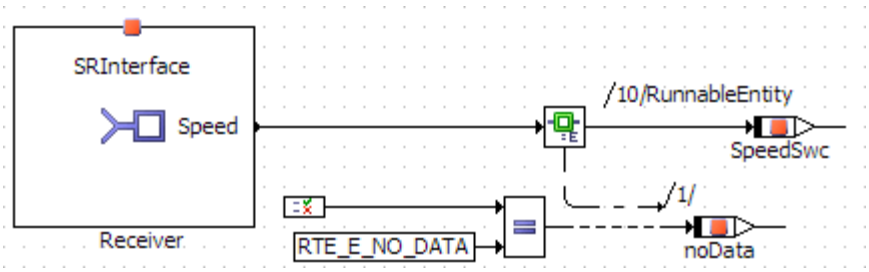
```
...
#define _Speed_SWC SWC_RAM.Speed_SWC
...
FUNC(void, SWC_CODE) RteRunnable_Swc_RunnableEntity (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   ...
   /* RunnableEntity: sequence call #10  */
   /* assignment to Speed_SWC: min=-32768, max=32767, hex=1phys+0,
   limit=(maxBitLength: true, assign: true), zero incl.=true  */
   _Speed_SWC = Rte_DRead_Receiver_Speed();
   ...
}
```

コード 103: C コード – 暗黙的受信（8.5.1 項「明示的な受信」– AUTOSAR R4.0.2）

ステータス付き明示的通信によるポートからの受信

明示的アクセスは、必要に応じてステータス付きで実装できます。ステータス付き明示的通信の設定については、10.3.1 項「ポートへの送信」（145 ページ）の「ステータス付き明示的通信によるポートへの送信」にある例を参照してください。

前項の例にステータス付き明示的通信を設定した場合、ASCET は以下のような C コードを生成します。



```
#define _noData Swc_RAM.noData
#define _SpeedSwc Swc_RAM.SpeedSwc

...

FUNC(void, CODE) RteRunnable_SWC_RunnableEntity (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;
   VAR(SInt16, AUTOMATIC) _tSpeed;

   ...
   /* RTE_ExplicitWithStatus-block: sequence call #Explicit write
   error/Status #1  */
   _ASCET_RteStatus = Rte_Read_Receiver_Speed(&(_tSpeed));
   if (_ASCET_RteStatus != RTE_E_OK)
   {
      _noData = _ASCET_RteStatus == RTE_E_NO_DATA;
   } /* end if */
   /* RunnableEntity: sequence call #10  */
   _SpeedSwc = _tSpeed;
}
```

コード 104: C コード – ステータス付きの明示的受信（AUTOSAR R3.1.2）

```
#define _noData SWC_RAM.noData
#define _Speed_SWC SWC_RAM.Speed_SWC
#define _Speed_SWC_REF_ (&(SWC_RAM.Speed_SWC))

...

FUNC(void, SWC_CODE) RteRunnable_Swc_RunnableEntity (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   ...
   /* RTE_ExplicitWithStatus-block: sequence call #Explicit write
   error/Status #1  */
   _ASCET_RteStatus = Rte_Read_Receiver_Speed(_Speed_SWC_REF_);
   if (_ASCET_RteStatus != RTE_E_OK)
   {
      _noData = _ASCET_RteStatus == RTE_E_NO_DATA;
   } /* end if */
}
```

コード 105: C コード – ステータス付きの明示的受信（AUTOSAR R4.0.2）

Rte_Read は読み取るデータがない場合でもノンブロッキングで実行されます。データがない場合、戻り値は RTE_E_NO_DATA になります。

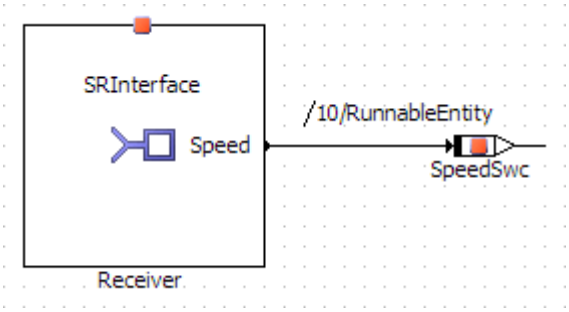
暗黙的通信によるポートからの受信

暗黙的 API の場合、データにアクセスするものとして宣言されているランナブルエンティティの参照が API 名に含まれています。ランナブルエンティティを作成する時には、正しい API を呼び出すようにするために注意が必要です。Rte_IRead API は以下のようにデータを読み取ります。

```
DataItemType Rte_IRead_<runnable>_<port>_<data>()
```

キャッシュはランナブルエンティティの起動前に更新されるので、ランナブルエンティティの 1 回の実行の中では、Rte_IRead から返される値は変わりません。

8.4.2 項「暗黙的な送信」に示されている例（下図）に対して、ASCET は以下のような C コードを生成します。




```
...
#define _SpeedSwc Swc_RAM.SpeedSwc
...
FUNC(void, CODE) RteRunnable_SWC_RunnableEntity (void)
{
   ...
   /* RunnableEntity: sequence call #10  */
   /* assignment to SpeedSwc: min=-32768, max=32767, hex=phys,
   limit=(maxBitLength: true, assign: true), zero incl.=true  */
   _SpeedSwc = Rte_IRead_RunnableEntity_Receiver_Speed();
}
```

コード 106: C コード - 暗黙的受信（8.4.2 項「暗黙的な送信」の例）

10.4 クライアント／サーバー通信

クライアント／サーバー通信は、 Rte_Call API コールにより開始されます。

CLIENT_MODE が同期モード（synchronous）に設定されている場合は、Rte_Call はサーバーによるオペレーションが完了してからコントロールを返します。つまり、ユーザーのコードは、サーバーが結果を返してくるまでは以降の処理を続行しません。結果は、算出されると Rte_Call の戻り値としてコンポーネントに返されます。

```
Rte_StatusType Rte_Call_<Port>_<Operation>(  InParam1Type  In_1,
…,
InParamNType  In_N,
OutParam1Type Out_1,
…,
OutParamMType Out_M)
```

10.4.1 サーバーオペレーションの実装

サーバーポートが定義されているコンポーネントには、オペレーション呼び出しイベントに対応するランナブルエンティティを実装する必要があります。そのようなランナブルエンティティのシグネチャは、10.2.2 項「ランナブルエンティティ用のエントリポイントシグネチャ」に明記されている規則に従う必要があります。

以下に、8.6 項「ポートへのサーバーリクエストへの応答」のランナブル `Server_MaximumValue` を実装する方法を紹介します。

サーバーオペレーションを実装する:

- 90 ページの「サーバーポートを作成する」の方法で、P ポート `Server` を作成します。
- ダイアグラム `Server_CSInterface` をロードします。
- 図 60 のようにオペレーション `Server_MaximumValue` を実装します。

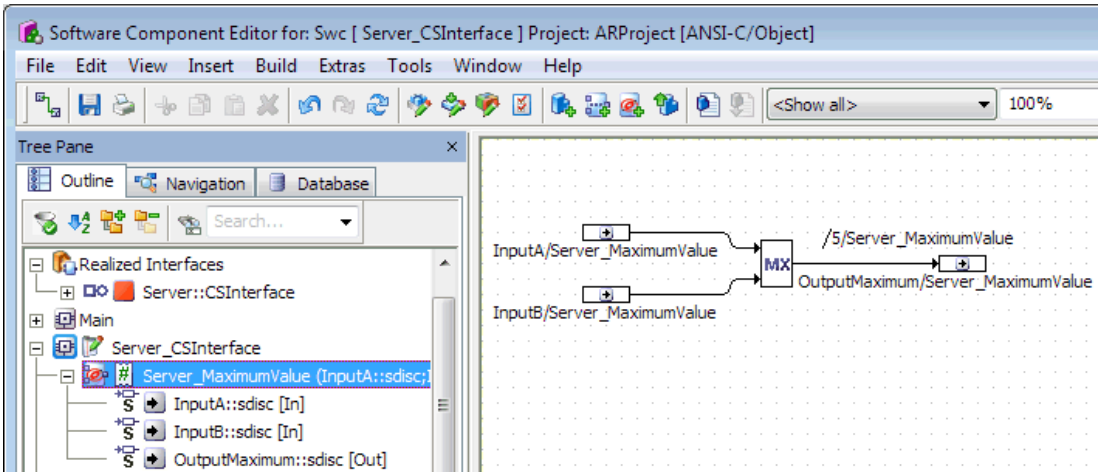


図 60: オペレーション `Server_MaximumValue` をダイアグラム `Server_CSInterface` に実装する

このオペレーション `Server_MaximumValue` について、ASCET は以下のようなサーバーランナブルを生成します。

```
/* begin region Runnable_Definition_Server_MaximumValue_ */
/*****************************************************************************
 * BEGIN: DEFINITION OF RUNNABLE 'Swc_Impl_Server_MaximumValue'
 * ---------------------------------------------------------------------------
 * model name:.................................'Server_MaximumValue'
 * memory class:...............................'CODE'
 * ---------------------------------------------------------------------------*/
/* messages used by this runnable */
/* public Server_MaximumValue
          (InputA::sdisc;InputB::sdisc;OutputMaximum::sdisc)  */

FUNC(void, CODE) Swc_Impl_Server_MaximumValue (
   /* IN    */ VAR(SInt16, AUTOMATIC)                         InputA,
   /* IN    */ VAR(SInt16, AUTOMATIC)                         InputB,
   /* OUT   */ CONSTP2VAR(SInt16, AUTOMATIC, RTE_APPL_DATA) OutputMaximum
   )
{
   /* Server_MaximumValue: sequence call #5  */
   /* assignment to OutputMaximum: min=-32768, max=32767, hex=phys,
   limit=(maxBitLength: true, assign: true), zero incl.=true  */
   (*OutputMaximum) = ((InputA >= InputB) ? InputA : InputB);
}
```

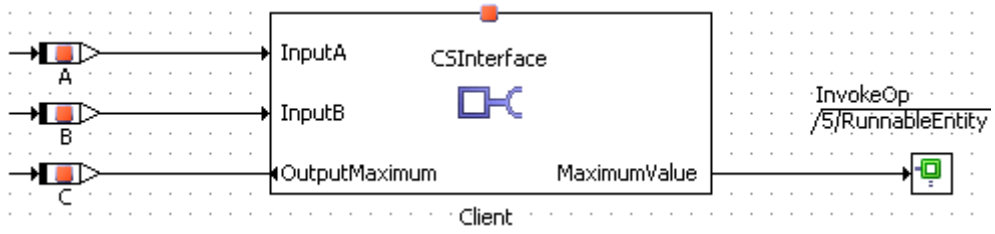
コード 107: C コード – サーバーランナブル

サーバーは、クライアントから通信サービス経由で受信したリクエストで呼び出されたり、タスク内通信で直接呼び出されたりするなど、複数のソースから呼び出される可能性があります。ランナブルのコンフィギュレーション内に「同時実行可能」としてマークされていないサーバーに対するリクエストは、RTE により先入れ先出し(FIFO)方式でキューイングされ、サーバーへのアクセスは逐次化されます。

10.4.2 ポートにクライアントリクエストを発行する

ランナブルエンティティは、サーバーポートにオペレーション要求が発行されるたびに RTE によって呼び出されます。

8.7 項「ポートへのクライアントリクエスト」(123 ページ)の例(下図)に対して、ASCET は以下のような C コードを生成します。



```
#define _A Swc_RAM.A
#define _B Swc_RAM.B
#define _C Swc_RAM.C
#define _C_REF_ (&(Swc_RAM.C))
...

FUNC(void, CODE) RteRunnable_SWC_RunnableEntity (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   ...
   /* RunnableEntity: sequence call #5  */
   _ASCET_RteStatus = Rte_Call_Client_MaximumValue(_A,_B,_C_REF_);
   ...
}
```

コード 108: C コード – クライアントリクエスト

10.5 適合パラメータへのアクセス

適合パラメータが宣言されているソフトウェアコンポーネントは、実行時に以下の API コールを使用して各特性値にアクセスします。

`CalprmElementType Rte_Calprm_<Port>_<CalprmElement>()`

このコールは適合データ(基本型)、またはデータのポインタ(複合型)を返します。

ファンクション内の適合データは ASCET の「パラメータ」を用いてモデリングされます。アプリケーションソフトウェアコンポーネント内では、パラメータマッピングテーブルを使用して適合データを AUTOSAR 適合コンポーネントの適合パラメータに対応付けることができます。

パラメータを含むファンクションを作成する:

- ASCET コンポーネントマネージャで **Insert → Class →Block diagram** を選択し、ASCET クラスを作成します。
- このクラスの名前を `ClassWithParam` にします。
- ブロックダイアグラムエディタで `ClassWithParam` を開きます。
-  **Logic Parameter** ボタンで論理パラメータを作成します。
  “Properties for Scalar Element: log”ダイアログボックスが開きます。
- このパラメータの名前を `localLog` にし、スコープを **Imported** に変更します。

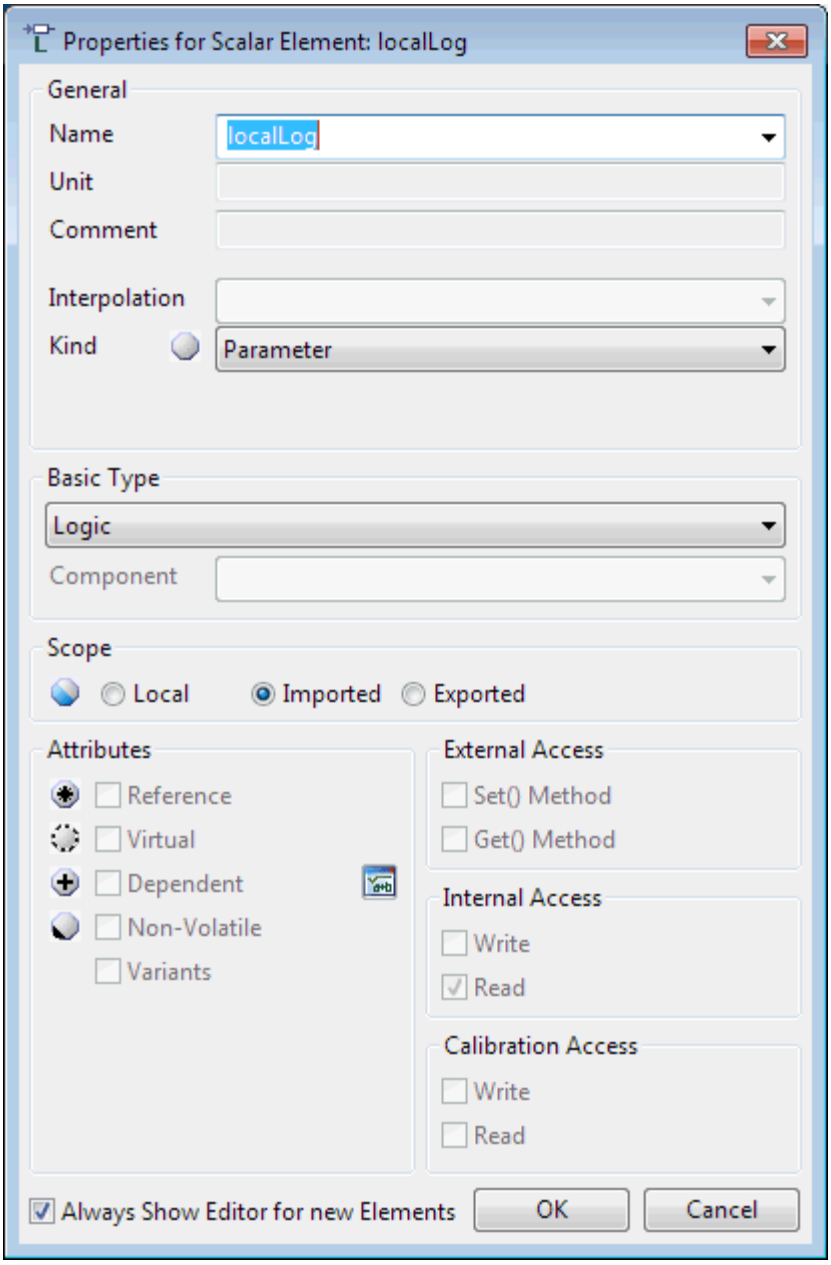


図 61: パラメータ `localLog` のスコープを `Imported` に設定する

- `localUdisc` という名前の符号なし離散パラメータを作成し、スコープを **Imported** に変更します。
- 以下のメソッドをモデリングします。

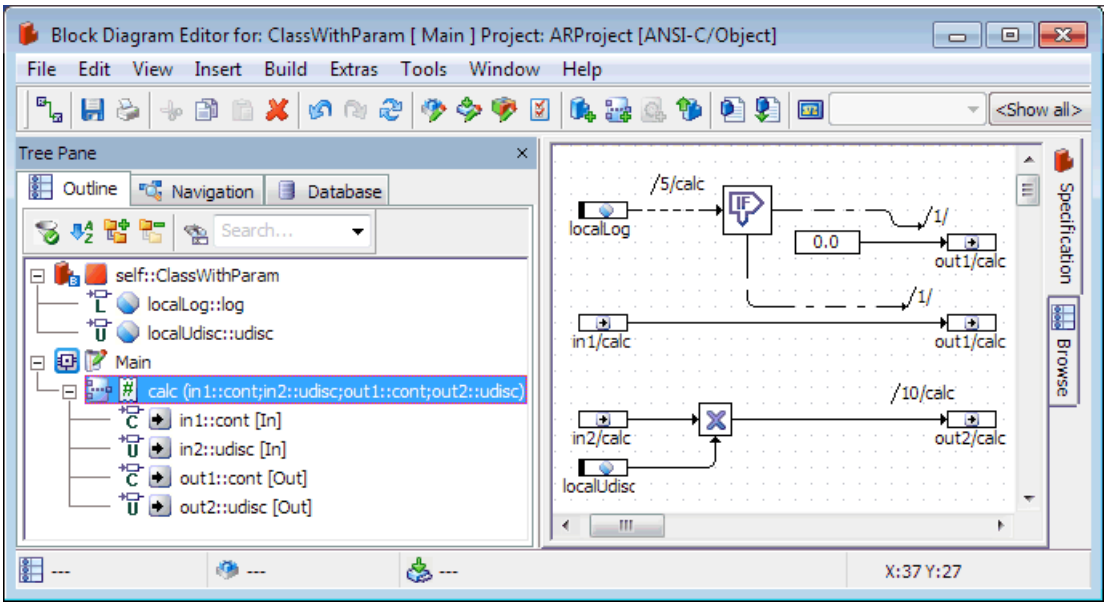


図 62: メソッド `calc` のブロックダイアグラム

ファンクションの内部パラメータを AUTOSAR 適合パラメータにマッピングする:

- 23 ページの方法でプロジェクトを作成します。
- 26 ページの方法で、新しいプロジェクトにソフトウェアコンポーネント `Swc` を挿入します。

  `Swc` に、78 ページの 6.4 項「適合」で作成した適合インターフェース `CalInterface` を挿入します。
- `Swc` に、クラス `ClassWithParam` を挿入します。
- 以下のブロックダイアグラムのように、変数 `inValue1`、`inValue2`、`outValue1`、`outValue2` を挿入します。

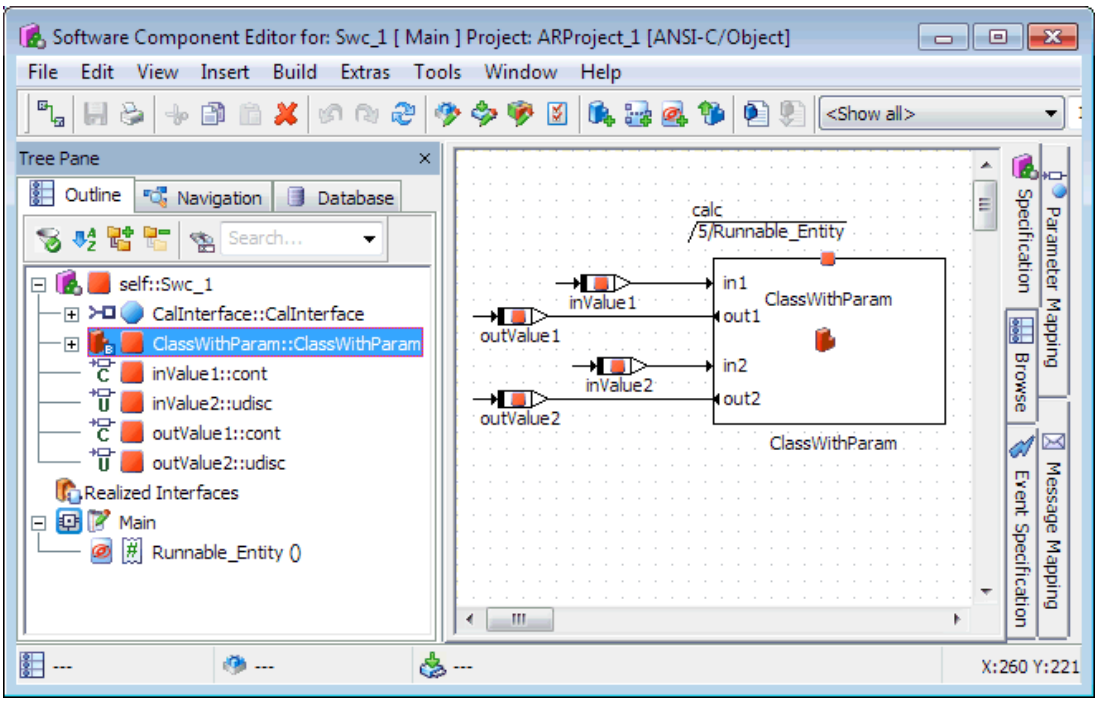


図 63: ソフトウェアコンポーネント内の `ClassWithParam` へのアクセス

- ランナブル `Runnable_Entity` を作成し、このランナブル内のメソッド `calc` にシーケンスを割り当てます。
- ソフトウェアコンポーネントエディタの“Parameter Mapping”タブを選択します。

  テーブルの左列には、ソフトウェアコンポーネントのモジュールとクラスにインポートされたすべてのパラメータがリストアップされます。

  右列は、各インポートパラメータのドロップダウンリストになっています。ソフトウェアコンポーネント内の適合パラメータのうち、左列のパラメータと型が一致するものが選択肢として提示されます。
- パラメータ `localLog` について、適合パラメータ `calParam1` を選択します。

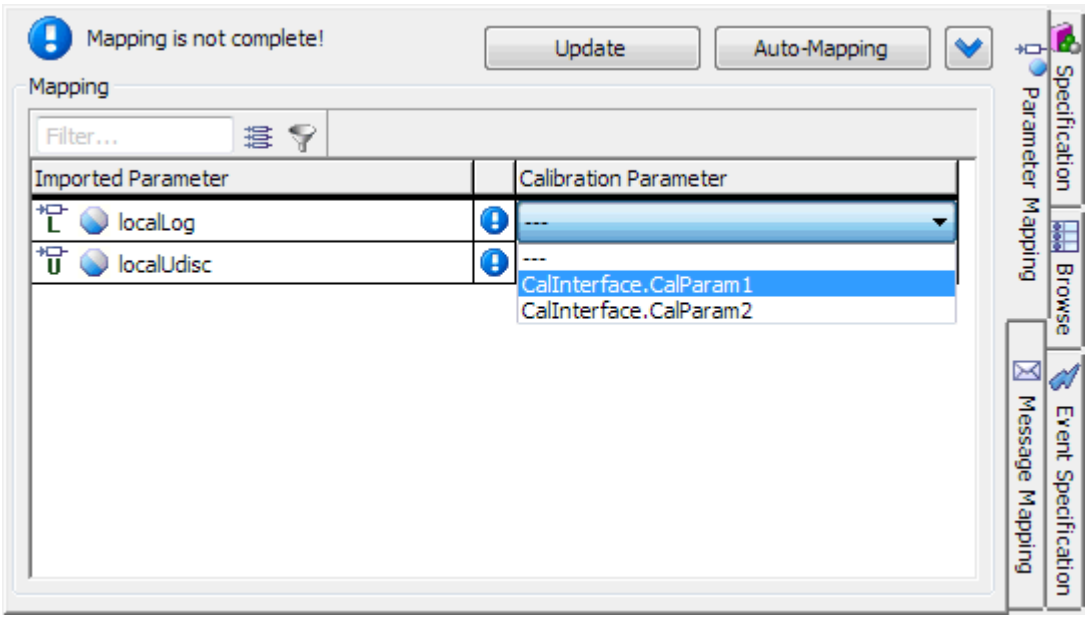
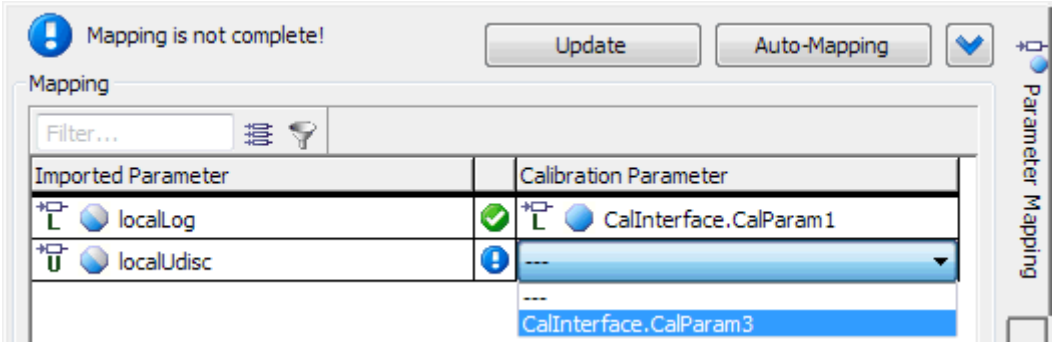


図 64: インポートパラメータと適合パラメータとのマッピング

- パラメータ `localUdisc` について、適合パラメータ `calParam3` を選択します。



これでパラメータのマッピングは終了です。

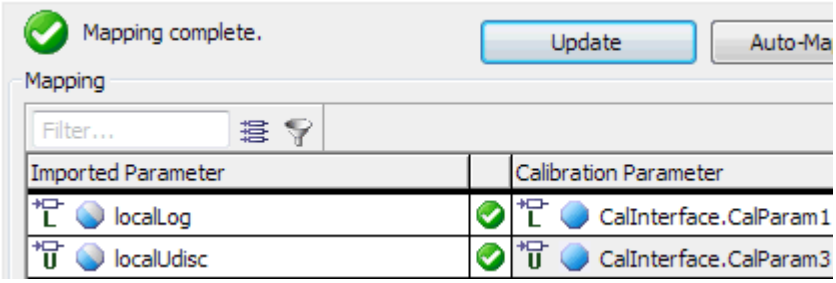


図 65: マッピングされたパラメータ

クラス ClassWithParam について、ASCET は以下のような C コードを生成します。

```
FUNC(void, CODE) CLASSWITHPARAM_IMPL_calc (
   /* IN     */ VAR(SInt32, AUTOMATIC)                        in1,
   /* IN     */ VAR(UInt8, AUTOMATIC)                         in2,
   /* OUT    */ CONSTP2VAR(SInt32, AUTOMATIC, RTE_APPL_DATA) out1,
   /* OUT    */ CONSTP2VAR(UInt8, AUTOMATIC, RTE_APPL_DATA)  out2
   )
{
   /* calc: sequence call #5  */
   if (Rte_Calprm_CalInterface_CalParam1())
   {
      /* If-block: sequence call #5/Then #1  */
      (*out1) = 0;
   }
   else
   {
      /* If-block: sequence call #5/Else #1  */

      (*out1) = in1;
   } /* end if */
   /* calc: sequence call #10  */
   (*out2) = (UInt8)((UInt16)in2 * Rte_Calprm_CalInterface_CalParam3());
}
```

コード 109: C コード – マッピングされたパラメータを含むクラス

注記

ソフトウェアコンポーネントが開いている時に適合インターフェースを編集した場合は、メニューコマンド **Mapping →Update** を実行して、“Parameter Mapping”タブでの変更内容を更新する必要があります。

10.6 ASCETメッセージへのアクセス

AUTOSAR には ASCET メッセージの概念がありません。作成した SWC が ASCET メッセージを含むモジュールを使用している場合、すべてのメッセージをセマンティック的に等価である AUTOSAR エレメントにマッピングする必要があります。

このマッピングは、ASCET のソフトウェアコンポーネントエディタの“Message Mapping”ビューで専用のエディタを使用して行うことができます。

このエディタでは、メッセージを以下の規則に従って AUTOSAR エレメントにマッピングすることができます。

- 各メッセージは互換性のある型のエレメントにのみマッピングできます。

メッセージの型	AUTOSARエレメントの型
Continuous (cont)	cont / sdisc / udisc
Signed Discrete (sdisc)	cont / sdisc / udisc
Unsigned Discrete (udisc)	cont / sdisc / udisc
Logic (log)	log
Enumeration (enum)	Enumeration（同じ型）

表 3: ASCET メッセージと互換性のある AUTOSAR の型

- 純粋な送信メッセージは、P ポートとして使用されるセンダ／レシーバインターフェースのエレメントにのみマッピングできます。これは、メッセージの値は SWC 内で使用されず、別の SWC が使用するためのものであるからです。

  「純粋な送信メッセージ」とは、SWC 内の 1 つのモジュールにのみ存在し、別のモジュールによって受信されないものを指します。
- 純粋な受信メッセージは、R ポートとして使用されるセンダ／レシーバインターフェースのエレメントにのみマッピングできます。これは、メッセージの値は SWC 内では与えられず、別の SWC によって与えられなければならないためです。

  「純粋な受信メッセージ」とは、SWC 内で送信メッセージとして使用されない受信メッセージを指します。
- 他のすべてのメッセージ（センダ／レシーバメッセージ、および 1 つのモジュールで送信メッセージとして定義され別のモジュールで受信されるメッセージ）は、インターランナブル変数または P ポートとして使用されるセンダ／レシーバインターフェースのエレメントにマッピングすることができます。

ASCET モジュールを SWC 内で容易に再利用できるようにするため、SWC からマッピングをエクスポートして別の SWC にインポートすることができます。詳しくは ASCET オンラインヘルプを参照してください。

メッセージを含むモジュールを作成する:

- ASCET コンポーネントマネージャで、**Insert → Module → Block diagram** を選択して ASCET モジュールを作成します。
- そのモジュールの名前を `ModuleWithMsg` にします。
- ブロックダイアグラムエディタで `ModuleWithMsg` を開きます。
-  **SendReceive Message** ボタンで受信メッセージを作成します。

  "Properties for Scalar Element: message"ダイアログボックスが開きます。
- メッセージの名前を `SendRecMsg1` にして、基本型を `Signed Discrete` に変更します。
- **OK** をクリックしてプロパティエディタを閉じます。
- 第 2 の送受信メッセージを追加し、名前を `SendRecMsg2` にして、基本型を `Signed Discrete` に変更します。
-  送信メッセージを追加し、名前を `SendMsg` にして、基本型を `Logic` に変更します。
- `SendRecMsg1` と `SendRecMsg2` を `sint8` として実装します（図 **10** 参照）。
- `SendMsg` を `bool` として実装します。
- 以下のプロセスをモデリングします。

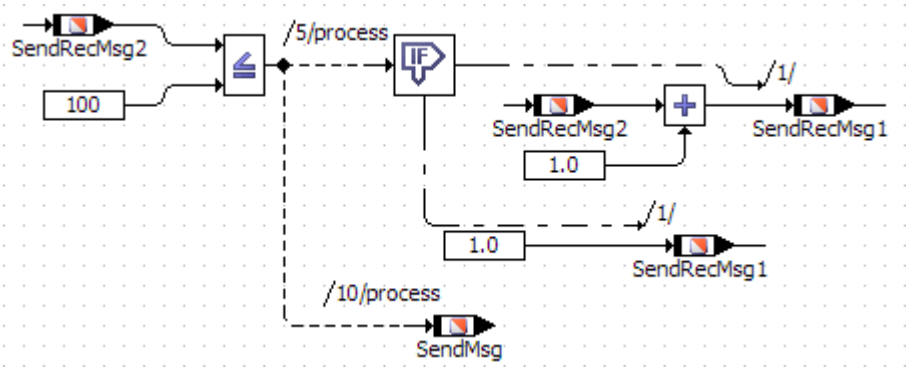


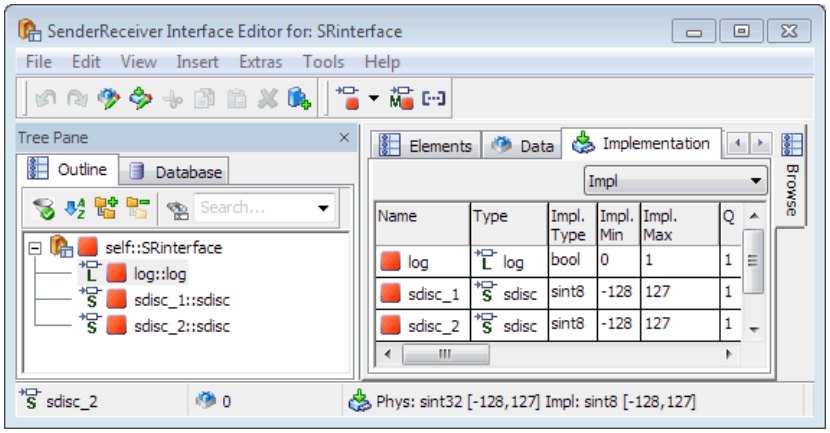
図 66: プロセス process のブロックダイアグラム

ASCET メッセージを AUTOSAR エレメントにマッピングする:

- プロジェクト ARProject を作成し(23 ページ参照)、セットアップします(23 ページ参照)。
- ソフトウェアコンポーネント Swc を作成し(26 ページ参照)、1 つのランナブルエンティティと 3 つのインターランナブル変数を挿入します(125 ページ参照)。

名前	IRV_sdisc1	IRV_sdisc2	IRV_log
基本型	Signed Discrete		Logic
実装型	sint8		bool
内部アクセス	**Implicit**	**Explicit**	**Implicit**

- センダ／レシーバインターフェース SRinterface(62 ページ参照)を作成し、sdisc データエレメントを 2 つ挿入して sint8 として実装し、さらに log エレメントを 1 つ挿入して bool として実装します。



- モジュール ModuleWithMsg を Swc に追加します。
- SRinterface を使用して、Swc 内にセンダポートを作成します(87 ページ参照)。
- Swc を ARProject に追加します。
- プロジェクトエディタウィンドウの“Outline”タブで Swc をダブルクリックし、コンポーネントをプロジェクトコンテキストとして開きます。
- ソフトウェアコンポーネントエディタの“Message Mapping”タブを選択し、さらに上部の“Internal Access”タブを選択します。

  テーブルの左列にはインターランナブル変数にマッピングできるすべてのメッセージがリストアップされます。

テーブルの右列は各メッセージ用のドロップダウンリストになっています。各リストにはメッセージにマッピングできるインターランナブル変数が表示されます。

- 図 67 のように、メッセージをインターランナブル変数にマッピングします。

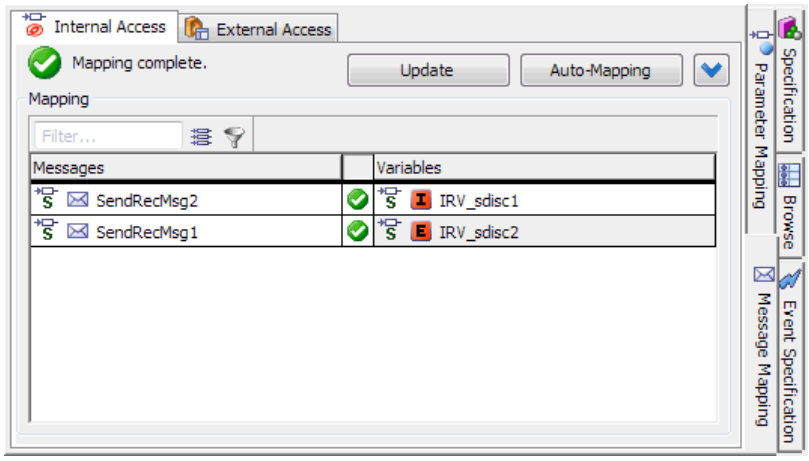


図 67: メッセージとインターランナブル変数のマッピング

- 上部の“External Access”タブを選択します。

  テーブルの左列には、センダポートまたはレシーバポートのデータエレメントにマッピングできるすべてのメッセージがリストアップされます。

  テーブルの右列は各メッセージ用のドロップダウンリストになっています。各リストにはメッセージにマッピングできるデータエレメントが表示されます。

- 図 68 のように、メッセージをデータエレメントにマッピングします。

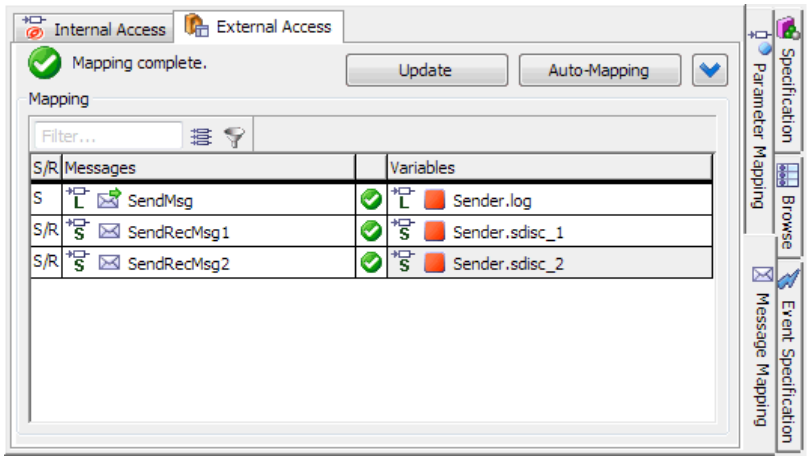


図 68: メッセージとデータエレメントのマッピング

これでメッセージマッピングが終了しました。

モジュール `ModuleWithMsg` について、ASCET は以下のような C コードを生成します。

```
FUNC(void, CODE) MODULEWITHMSG_IMPL_process (void)
{
   /* temp. variables */
   VAR(sint8, AUTOMATIC) _t1sint8;
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;

   /* process: sequence call #5  */
   if (Rte_IrvIRead_runnable_IRV_sdisc1() <= 100)
   {
      /* If-block: sequence call #5/Then #1  */
      _t1sint8 = Rte_IrvIRead_runnable_IRV_sdisc1();
      _ASCET_RteStatus
         = Rte_Write_Sender_sdisc_1(((_t1sint8 <= 126) ? (_t1sint8 + 1) : 127));
      _t1sint8 = Rte_IrvIRead_runnable_IRV_sdisc1();

      Rte_IrvWrite_runnable_IRV_sdisc2(((_t1sint8 <= 126) ? (_t1sint8 + 1) : 127));
   }
   else
   {
      /* If-block: sequence call #5/Else #1  */
      _ASCET_RteStatus = Rte_Write_Sender_sdisc_1(1);

      Rte_IrvWrite_runnable_IRV_sdisc2(1);
   } /* end if */
   /* process: sequence call #10  */

   Rte_IWrite_runnable_Sender_log(Rte_IrvIRead_runnable_IRV_sdisc1() <= 100);
}
```

コード 110: C コード – マッピングされたメッセージを含むモジュール

10.7 排他領域による並行処理制御

1 つのコンポーネントの中に、同じプロトタイプステートへの同時書き込みアクセスを必要とするランナブルエンティティが 2 つ以上ある場合、API `Rte_Enter` および `Rte_Exit` を使用してデータの整合性を確実に維持する必要があります。

1 つのコンポーネントには、同時に起動きる複数のランナブルエンティティが定義されているので、これらのランナブルエンティティがプライベートグローバルデータ（データメモリセクション内のエレメントなど）や非リエントラントな関数に同時にアクセスしてしまう可能性があります。

OS の並行処理制御メカニズムはコンポーネントからは隠されていますが、以下の RTE API を用いることによって適切な OS メカニズムをコンポーネントに適用し、排他領域への明示的アクセスを実現することができます。

- `Rte_Enter_<exclusive area name>` enters an exclusive area.
- `Rte_Exit_<exclusive area name>` exits an exclusive area.

排他領域を宣言しているコンポーネントのために生成される RTE API にはこれらの API コールが含まれるので、共有データへの同時アクセスを制御できます。

10.7.1 排他領域に割り当てられるランナブルのシーケンス

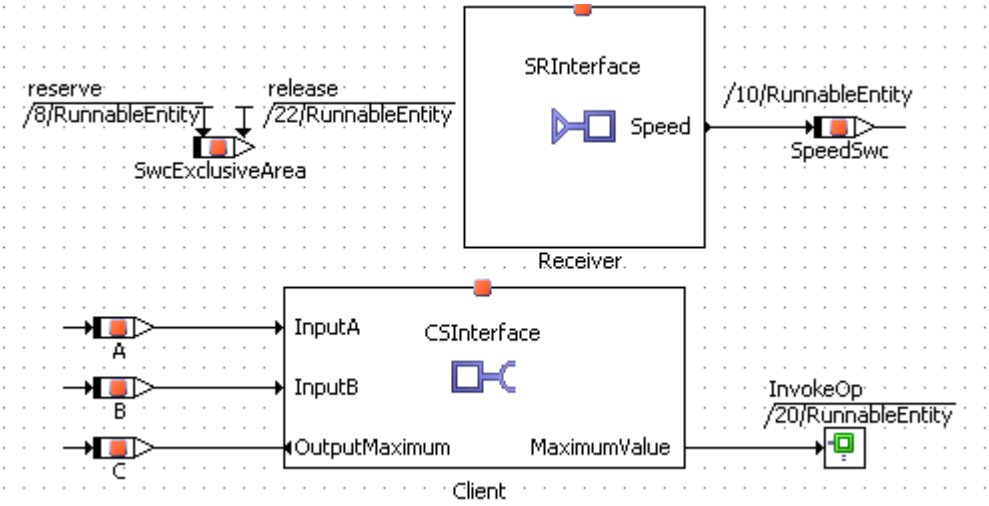
コンポーネントでは、ユーザーが設定時に定義した任意の排他領域 ID について、API コール `Rte_Enter` および `Rte_Exit` を使用できます。

8.9 項 8.9「排他領域」（129 ページ）で作成した排他領域 `SwcExclusiveArea` には、以下のようなコールが使用されます。

```
Rte_Enter_SwcExclusiveArea();
/* Code protected from concurrent execution */
...
Rte_Exit_SwcExclusiveArea();
```

コード 111: C コード – 排他的領域の入口と出口

131 ページの 8.9.3 項の例（下図）に対して、ASCET は以下のような C コードを生成します。




```
#define _A SWC_RAM.A
#define _B SWC_RAM.B
#define _C_SWC RAM.C
...

FUNC(void, CODE) RteRunnable_Swc_RunnableEntity (void)
{
   /* temp. variables */
   VAR(Std_ReturnType, AUTOMATIC) _ASCET_RteStatus;
   ...
   /* RunnableEntity: sequence call #8  */
   Rte_Enter_SwcExclusiveArea();
   /* RunnableEntity: sequence call #10  */
   _Speed_SWC = Rte_IRead_RunnableEntity_Receiver_Speed();
   /* RunnableEntity: sequence call #20  */
   _ASCET_RteStatus = Rte_Call_Client_MaximumValue(_A,_B,_C_REF_);
   /* RunnableEntity: sequence call #22  */
   Rte_Exit_SwcExclusiveArea();
}
```

コード 112: C コード – 排他的領域のコード例

注記

排他領域のスコープは「ソフトウェアコンポーネント**プロトタイプ**」であり、ソフトウェアコンポーネント型やシステム全体ではないので、排他領域で並行処理を制御できるのは 1 つのソフトウェアコンポーネントの中だけです。ただし AUTOSAR コンポーネントを使用して共有データへのアクセスの調停を行うと、これより広いスコープを実現することができます。

11 お問い合わせ先

ETAS 本社

ETAS GmbH
Borsigstraße 14 Phone: +49 711 89661-0
70469 Stuttgart Fax: +49 711 89661-106
Germany WWW: www.etas.com/

ETAS の各支社と技術サポート

各地域の支社と技術サポート窓口につきましては、ETAS ホームページをご覧ください。

各地域の支社 WWW: www.etas.com/en/contact.php
技術サポート WWW: www.etas.com/en/hotlines.php

索引