ASCET V5.2

入門ガイド

著作権について

Document EC010010 R5.2.2 JP

目次

1 はじめに

ASCET は、組み込みソフトウェアシステムのファンクション開発とソフトウェア開発のための革新的なソリューションを提供します。ASCET は、新しい独自のアプローチによって、モデリング、コード生成、シミュレーション実験、といった開発工程の各段階を強力にサポートするので、品質向上や開発サイクルの短縮、さらにコスト低減を実現できます。

本書は、読者の方が ASCET について理解し、速やかに成果を得られるように支援するものです。システムについて順を追って紹介すると同時に、すべての情報を参考資料として利用しやすい形にまとめてあります。

1.1 ASCET ファミリ

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。最新バージョンにおいては、ASCET 製品ファミリは以下の製品で構成されています。

- **ASCET-MD**
  モデルの開発とオフラインシミュレーションに使用します。
- **ASCET-RP**
  実験ターゲット（マイクロコントローラを搭載した ETAS シミュレーションボード）を使用した HiL（Hardware-in-the-Loop）シミュレーションやラピッドプロトタイピングをサポートします。また INTECRIO との接続機能も含まれます。
- **ASCET-SE**
  ECU の各種マイクロコントローラターゲットへのコード実装をサポートします。ターゲットとなるマイクロコントローラ用に最適化された実行コード（任意に設定されたオペレーティングシステムを含む）が生成されます。さまざまなタイプのコントローラをサポートし、2 種類の OS を統合できます。

また、以下のオプションモジュールも利用できます。

- **Configuration management**
  外部のコンフィギュレーション管理ツールとのインターフェースを提供します。
- **ASCET-MIP**
  このモジュールには、MATLAB ソフトウェアにアクセスするための 2 つの機能が含まれています。1 つは MABLAB エンジンへのインターフェースで、これによって ASCET と MATLAB をシミュレーションレベルで結合することができます。もう 1 つは、Simulink モデルを ASCET に読み込むためのモデルコンバータです。
- **ASCET-DIFF**
  ASCET モデルの比較ツールです。

これらの他にも、ご要望に応じてユーザー固有の製品を ASCET に組み込むことも可能です。詳しくは ETAS までお問い合わせください。

1.2 マニュアルについて

1.2.1 ユーザープロファイル

このマニュアルは、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。このマニュアル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、Windows 2000 または Winidows XP の操作についての知識も必要です。メニューコマンドの実行、ボタン操作、さらには Windows のファイルシステム、特にファイルとディレクトリの関係についての知識が必要です。また Windows のファイルマネージャやエクスプローラ等の基本的な使い方を理解されていて、「ドラグアンドドロップ」操作に慣れていることも必要です。

Microsoft Windows の基本テクニックに慣れていない方は、まずそれらについて学習してから、ASCET をご使用ください。Windows オペレーティングシステムについての詳しい説明は、マイクロソフト社発行のマニュアルを参照してください。

またさらに ANSI C や JAVA 等のプログラミング言語に関する知識があれば、ASCET をより効率的にお使いいただくことができます。

1.2.2 マニュアルの構成

ASCET マニュアルは、以下の 3 つのドキュメントで構成されています。

1. 『ASCET V5.2 入門ガイド』
   ASCET で作業する際に必要な基本的な情報がまとめられています。
2. 『ASCET V5.2 ユーザーズガイド』
   ASCET-MD の使用方法について詳しく説明されています。
3. 『ASCET V5.2 リファレンスガイド』
   ASCET モデリング言語についての詳しい解説や、その他の補足情報がまとめられています。

各ドキュメントの内容は、以下のとおりです。

『ASCET V5.2 入門ガイド』

- **「はじめに」**（本章）
  ASCET 製品およびマニュアルについての概要がまとめられています。
- **「プログラムのインストール」**
  この章は、PC またはネットワーク上に ASCET をインストールし、ASCET プログラムのメンテナンスやアンインストールを行うすべてのユーザー、および ASCET をファイルサーバに供給してネットワーク経由のインストールを行うシステム管理者を対象としています。ここには、製品の内

容、スタンドアロンインスタレーションとネットワークインスタレーションに必要なハードウェアやソフトウェアの要件、およびインストールの準備作業、さらにインストールとアンインストールの手順も紹介されています。

- **「ASCET を理解する」**

  ASCET システムの機能とその応用分野、および ETAS ツールチェーンにおける ASCET の役割りなどについてまとめられています。ASCET を初めて使用する方は、必ずお読みください。

- **「ライセンスについて」**

  ASCET を使用するためのライセンスについての情報（ライセンスファイルの入手方法、ライセンスの借用方法など）が説明されています。

- **「ASCET の基本操作」**

  ASCET のユーザーインターフェース（ウィンドウやメニュー）の機能についての一般的な情報や、マウスやキーボードによる操作方法がまとめられています。

- **「チュートリアル」**

  この章は、ASCET を初めて使うユーザーを対象としています。例題について実際に作業を進めながら、ASCET の使用方法を学習できます。チュートリアルは各工程ごとにいくつかのセクションに分かれていて、各セクションは互いに関連し合っています。なお、本章を読む際は、あらかじめ「ASCET を理解する」（49 ページ）を読んで ASCET の基本概念を理解しておいてください。

注記

ETAS では、ASCET のユーザートレーニングを実施しています。トレーニングにご参加いただくことにより、ASCET の基本的な機能と使用方法のほか、より詳しい知識を短時間で習得していただくことができます。

- **「用語集」**

  マニュアルで使用されている技術用語について解説されています。

- **「参考資料」**

  トラブルシューティングやディレクトリ構成についての情報、およびキーボードショートカットの一覧がまとめられています。

『ASCET V5.2 ユーザーズガイド』

このマニュアルには、ASCET システムに含まれるすべてのコンポーネントとその使用方法について、詳しく説明されています。このマニュアルを読むには ASCET についての基礎的な知識が必要ですので、ASCET を初めて使用する場合は、先に『ASCET V5.2 入門ガイド』の「ASCET の基本操作」および「チュートリアル」の項をお読みいただくことをお勧めします。

ASCET システムの操作方法が、ASCET で実際に組み込み制御システムを開発する際の手順に沿って説明されています。

このマニュアルは以下の章で構成されています。

- **「はじめに」**

  典型的なワークフローについての説明です。

- **「コンポーネントマネージャ」**

  コンポーネントマネージャのユーザーインターフェースについて、機能と操作方法、さらにそのカスタマイズの方法についても説明されています。

  この章は、ASCET を使用するすべてのユーザーを対象としています。

- **「ユーザー定義機能の追加」**

  ASCET の各ウィンドウに独自のメニューコマンドを追加する方法について説明されています。

- **「コンポーネントとプロジェクトの定義」**

  各種エディタの使用方法や、エレメント属性の変更方法などについて説明されています。

- **「シグナルとアイコン」**

  データベース内に測定信号やアイコンを取り込んで使用する方法について説明されています。

- **「実験」**

  オフライン実験環境の使用方法と、実験環境で使用される測定／適合ウィンドウの概要について説明されています。

  オンライン実験や、INCA や INTECRIO での実験については、『ASCET-RP ユーザーズガイド』を参照してください。

- **「ドキュメントの自動生成」**

  ASCET のコンポーネントやプロジェクトについてのドキュメントを自動生成する方法について説明されています。

『ASCET V5.2 リファレンスガイド』

このドキュメントの前半の「モデリング言語」の部分には、総合的なリファレンス情報として、ASCET で組み込みソフトウェアシステムを記述する手法が記載されています。ASCET を初めて使うユーザーは、この部分を読む前に『ASCET V5.2 ユーザーズガイド』のチュートリアルの課題をすべて終了させておくことをお勧めします。

リファレンスガイドは、以下の章で構成されています。

- **「プロジェクト」**

  組み込み制御システムについて記述されたものをプロジェクトと呼びます。ここでは、プロジェクトの構造について説明します。

- **「コンポーネント」**

  コンポーネントは組み込みシステムを構成する基本単位です。この章では、各種のコンポーネントが紹介されています。

- **「型とエレメント」**

  ASCET がサポートする変数の種類とデータ型が紹介されています。

- **「データとインプリメンテーション」**

  ASCET の各変数には、ここで説明するデータとインプリメンテーションが定義されます。

- **「ESDL によるボディ記述」**

  ASCET のモデル記述言語「ESDL」によるコンポーネントの記述方法について説明されています。

- **「ブロックダイアグラムによるボディ記述」**

  コンポーネントをブロックダイアグラムで記述する方法について説明されています。

- **「C によるボディ記述」**

  コンポーネントは C 言語で記述することもできます。その方法は、この章で説明されています。

- **「連続系」**

  ASCET における連続系のモデリングについての概要です。

- **「連続系基本ブロック」**

  連続系に使用されるコンポーネントについて詳しく記述されています。

- **「連続系構造ブロックと階層的グラフィック記述」**

  基本ブロックを完全なモデルに統合するために使用される構造ブロックについての説明です。

- **「プロジェクトとハイブリッドプロジェクト」**

  連続系を組み込みシステムと並行に実行する「ハイブリッドプロジェクト」について説明されています。

後半の「リファレンス」の部分には、ASCET のシステムライブラリ、およびその他の参考情報がまとめられています。

この部分は、以下の章で構成されています。

- **「ASCET システムライブラリ」**

  システムライブラリの各コンポーネントについての詳しい説明です。

- **「トラブルシューティング」**

  一般的なエラーや確認されている問題点と、その解決方法についてのアドバイスがまとめられています。

- **「コード生成時に出力されるメッセージ」**

  コード生成中に出力される可能性のあるエラーメッセージと、モデル記述を修正してそれらのエラーを回避する方法についてのアドバイスが紹介されています。

1.2.3 本書の使用法

表現について

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。

目標の定義：

前置き ...

- 手順 1

  手順 1 についての説明 ...
- 手順 2

  手順 2 についての説明 ...
- 手順 3

  手順 3 についての説明 ...

まとめ ...

具体例：

新しいファイルを作成する：

新しいファイルを作成する際は、他のファイルをすべて閉じておきます。

- **File → New** を選択します。

  “Create file”ダイアログボックスが開きます。
- 新しいファイルの名前を、“File name”フィールドに入力します。

  ファイル名は 8 文字以内でなければなりません。
- **OK** をクリックします。

新しいファイルが作成され、ユーザーが指定した名前で保存されます。このファイルを使用して以降の操作を行います。

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。

表記例	説明
File → Exit を選択して、...	メニューコマンドは、**青の太字**で表記します。
OK をクリックして、...	ユーザーインターフェース上のボタン名は、**青の太字**で表記します。
<Ctrl> を押して、...	キーボードの各キーは、**< >** で囲んで表記します。
“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` フォントで表記します。
論理型のデータから算術型のデータへの変換は**できません**。	注意すべき個所、または新出の用語は**太字**、あるいは「 」で囲んで表記されます。
OSEK グループ（http://www.osek-vdx.org/ を参照してください）はさまざまな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、青い下線で表記されています。

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

また PDF 文書において、索引、および他の部分を参照する個所（例：「xx を参照してください」の中の「xx」の部分）については、その参照先へのリンクが設けられているので、必要な参照個所を素早く見つけることができます。

2 プログラムのインストール

この章は、PC またはネットワーク上に ASCET をインストールし、ASCET プログラムのメンテナンスやアンインストールを行うすべてのユーザー、および ASCET をファイルサーバに供給してネットワーク経由のインストールを行うシステム管理者を対象としています。ここには、製品の内容、スタンドアロンインスタレーションとネットワークインスタレーションに必要なハードウェアやソフトウェアの条件、およびインストールに必要な準備について、重要な情報が記載されています。また、ASCET のインストールとアンインストールの手順も紹介しています。

2.1 準備

インストールを行う際は、まず製品の内容に不足がないこと、またお手持ちのコンピュータがシステム要件を満たしていることを確認してください。使用するオペレーティングシステムとネットワーク接続によっては、インストールを行うために必要なユーザー特権を持っていることを確認する必要があります。

2.1.1 製品の内容

ASCET は、以下のアイテムで構成されます。

- ASCET CD- ROM
  - ASCET のインストールプログラム
  - ASCET マニュアルおよび ETAS ハードウェアのドキュメント（PDF ファイル、開くためには Acrobat Reader が必要です）
  - エンドユーザー向け FLEXnet ライセンスユーザーズガイド（PDF ファイル）
  - Acrobat Reader のインストールプログラム

2.1.2 システム要件

ASCET を使用する PC は、以下の条件を満足している必要があります。

- 1GHz Pentium PC （2GHz 以上を推奨）
- Windows® 2000、Windows® XP
- 512MB RAM
- 空き容量が 1GB（プログラムデータ用サイズを含まず）以上のハードディスク
- CD ROM ドライブ
- VGA グラフィックカードと VGA モニタ、256 色、800 × 600 以上の解像度

2.1.3 インストールと運用に必要なユーザー特権

インストールに必要なユーザー特権：

ASCET を PC にインストールするには、管理者のユーザー特権が必要です。ない場合は、システム管理者の方にお問い合わせください。

運用に必要なユーザー特権：

ASCET を Windows 2000/XP で運用するには、各ユーザーが管理者から“スケジューリング優先順位の繰り上げ”という特権を受ける必要があります。この特権は、ユーザーマネージャーを使用して設定します。

注記

下記の設定を行うためには、管理者権限が必要です。

推奨：一般的には、以下のようにして“スケジューリング優先順位の繰り上げ”特権をローカルの“Users”グループに設定してください。

Win 2000 でユーザー特権“スケジューリング優先順位の繰り上げ”を割り当てる：

- Windows のスタートメニューから、**設定** → **コントロールパネル** → **管理ツール** → **ローカルセキュリティポリシー** を選択します。
- **ローカルポリシー** → **ユーザー権利の割り当て** に含まれる“スケジューリング優先順位の繰り上げ”をダブルクリックします。
- **追加** ボタンをクリックします。
- ローカルコンピュータを選択します。
- “Users”グループをダブルクリックして“スケジューリング優先順位の繰り上げ”を割り当てます。
- **OK** ボタンをクリックして確定します。
- **OK** ボタンで“ローカルセキュリティポリシーの設定”ダイアログボックスを閉じます。
- ローカルセキュリティ設定を終了します。

Win XP でユーザー特権“スケジューリング優先順位の繰り上げ”を割り当てる：

- Windows のスタートメニューから、**コントロールパネル** → **管理ツール** → **ローカルセキュリティポリシー** を選択します。

- **ローカルポリシー → ユーザー権利の割り当て** に含まれる“スケジューリング優先順位の繰り上げ”をダブルクリックします。

  “スケジューリング優先順位の繰り上げ”ダイアログボックスが開きます。
- **ユーザーまたはグループの追加** ボタンをクリックします。

  “ユーザーまたはグループの選択”ダイアログボックスが開きます。
- **場所** ボタンをクリックします。

  “場所”ダイアログボックスが開きます。
- ローカルコンピュータを選択し、**OK** をクリックして“場所”ダイアログボックスを閉じます。
- “ユーザーまたはグループの選択”ダイアログボックスで、**詳細設定** ボタンをクリックして、自動検索機能を有効にします。
- **今すぐ検索** ボタンをクリックして、ローカルコンピュータに登録されているユーザーのリストを表示します。
- “名前（RDN）”のカラムから、スケジューリング優先度の繰り上げの権利を割り当てたいユーザーまたはグループを選択します。
- **OK** ボタンをクリックして確定します。
- **OK** ボタンで“ユーザーまたはグループの選択”ダイアログボックスを閉じます。
- **OK** ボタンで“スケジューリング優先順位の繰り上げ”ダイアログボックスを閉じます。
- ローカルセキュリティ設定を終了します。

2.2 インストール

ASCET-MD、ASCET-RP、ASCET-SE（79 ページの 4.4 項を参照してください）のいずれの製品をインストールする場合も、まず最初に ASCET の基本システムをインストールする必要があります。ここでは、基本システムと ASCET-MD のインストールについて説明します。ASCET-RP と ASCET-SE のインストールについては、各製品のマニュアルを参照してください。

インストールの方法は、CD から行う場合もネットワークドライブから行う場合も同様です。

特殊なインストール条件（インストールのキャンセル、既存のプログラムバージョンへの上書き等）については、27 ページの 2.2.2 に説明されています。

2.2.1 初回のインストール

ASCET 基本システム

ASCET のインストールを開始する：

- スタートメニューから、**ファイル名を指定して実行**を選択します。
- コマンド行に、インスタレーションファイルのパスを入力します（たとえば、CD からインストールする場合には、`d:¥ASCET5.2¥ASCET.exe` と入力します）。
- **OK** をクリックして確定します。

  インスタレーションプログラムが起動します。

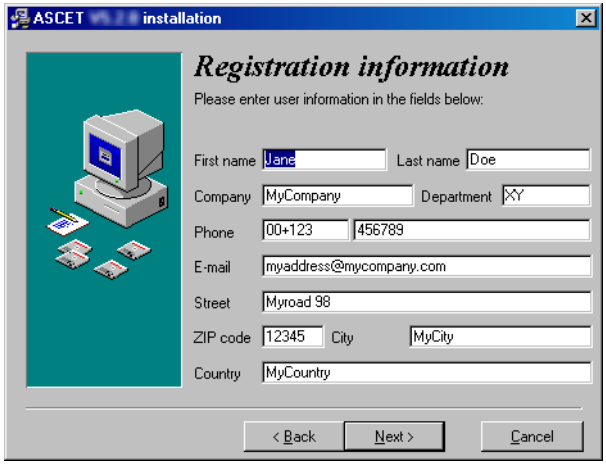
ライセンス契約の内容を確認する：

- “EULA”というタイトルのダイアログボックスが開き、ライセンス契約の内容が表示されるので、内容に同意いただける場合は、**Accept** オプションをクリックしてオンにしてください。
- **OK** をクリックして確定します。
- 画面に表示される指示に従って先に進みます。

  ダイアログボックス内に設定した内容を確定するには、**Next** ボタンをクリックして次のダイアログボックスに進みます。**Back** ボタンをクリックすると前のダイアログボックスに戻り、また **Cancel** をクリックするとインストールが中止されます。

ASCET に登録する：

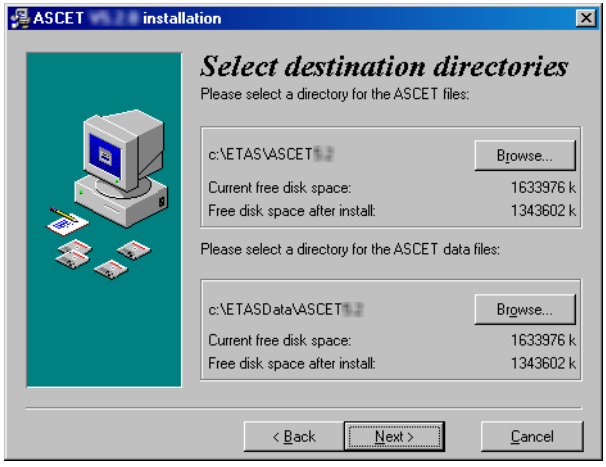
- 以下のダイアログボックスに、ユーザーの個人情報を入力します。



- **Next** ボタンをクリックします。

ASCET のパスを指定する：

ASCET の登録が終わると、インストール先のターゲットディレクトリを指定するように要求されます。この指定は、2 つのダイアログボックスで行います。



プログラムファイルとプログラムデータは別のディレクトリに格納されます。後でプログラムをアンインストールしたり更新すると、プログラムファイルだけが削除または上書きされます。プログラムデータはそのまま残り、継続して使用可能です。プログラムデータには以下のデータが含まれます。

- データベース
- ユーザープロファイル

注記

ASCET のインストールディレクトリのパスには、スペース文字を含めることができます。ただしその際は、ASCET と共に使用する外部ツールも空白文字を含むパス名をサポートしている必要がありますので、確認してからインストールを行ってください。

- デフォルトのディレクトリを変更したい場合には、**Browse** ボタンをクリックします。
- ダイアログボックスから、希望のディレクトリを選択します。

  存在しないディレクトリを指定すると、インスタレーションルーチンがそのディレクトリを自動的に作成します。
- **Next** ボタンをクリックします。

  “Select global directories”ダイアログボックスが開きます。ここでログファイルは一時ファイルの保存先を指定します。
- デフォルトのディレクトリを変更したい場合には、**Browse** ボタンをクリックします。
- **Next** ボタンをクリックします。

共有ファイルの設定を行う：

“Select Handling of ETAS Shared modules”ダイアログボックスで、ETAS の全製品で使用する基本モジュールの共有に関する設定を行います。

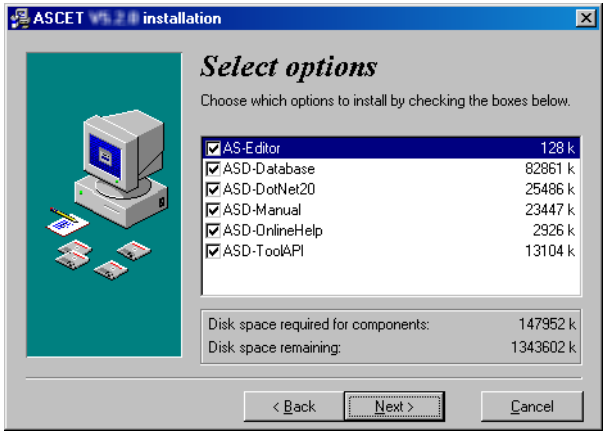
- モジュールを他の製品と共有して使用するには、**share modules between products** オプションをオンにします。

  複数の ETAS 製品を同時に使用する場合は、こちらの設定をお勧めします。
- モジュールのコピーをインストールして ASCET 専用で使用するには、**use local copies for each product** オプションをオンにします。各コンポーネントを個別に調整する必要がある場合は、こちらの設定にしてください。

- デフォルトのディレクトリを変更したい場合には、**Browse** ボタンをクリックします。
- **Next** ボタンをクリックします。

インストールする機能を指定する：

“Select options”ダイアログボックスで、ASCET の機能範囲を指定します。



- インストールしたいモジュールをチェックします。

  以下のオプションを選択できます。

  — “AS-Editor” - 算術演算サービスエディタ

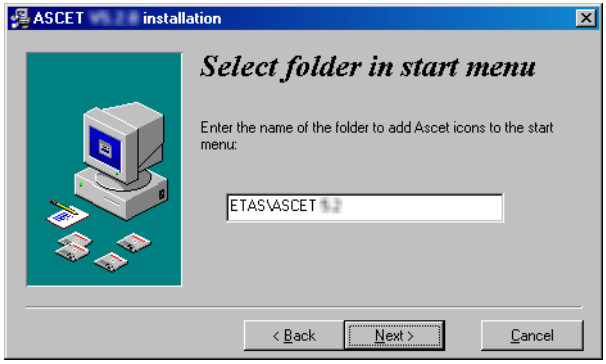
  — “ASD-Database” - ETAS システムライブラリとチュートリアルデータベースを、ASCET のインストールディレクトリにエクスポートします。

  — “ASD-Manual” - ASCET マニュアル（PDF）を `ETAS¥ETASManuals` ディレクトリにインストールします。

  — “ASD-OnlineHelp” - ASCET オンラインヘルプを `ETAS¥ASCET5.2¥help` ディレクトリにインストールします。

  — “ASD-ToolAPI” - ASCET の Automation インターフェースをインストールします。
- **Next** ボタンをクリックします。

スタートメニューに表示される ASCET フォルダ名を指定する：



- デフォルトのフォルダ名を有効にします。

または

- 別のフォルダ名を指定します。
- **Next** ボタンをクリックします。

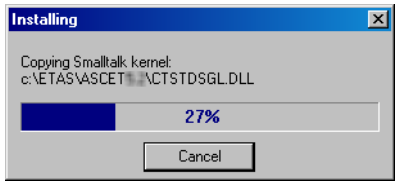
ASCET をインストールする：

注記

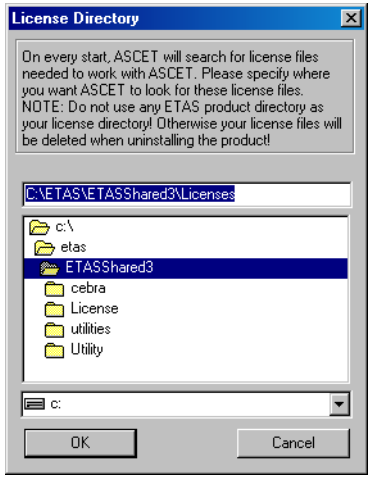
次のステップで、実際のインストール処理が開始されます。

- “Ready to Install”ダイアログボックスで **Next** をクリックして、インストールを開始します。

プログラムファイルがコピーされます。コピーの進捗状況は、棒グラフによって表されます。



すべてのコピーが終了すると、”License Directory”ダイアログボックスが開きます。



ライセンスファイルの格納場所を指定する：

- "License Directory”ダイアログボックスで、ライセンスファイルを保存する場所を指定します。

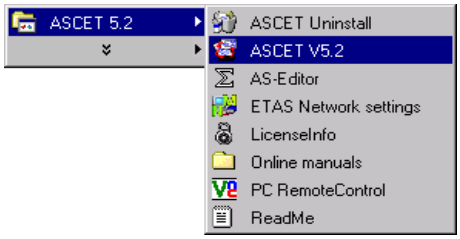
  ASCET を起動してさまざまな操作を行うと、適宜ライセンスファイルがチェックされますが、その際、このディレクトリからファイルが検索されます。

- **OK** をクリックして確定します。

  インストール処理が続行され、“Installation completed”というダイアログボックスが開きます。

- “Installation complete”ダイアログボックスで **Finish** ボタンをクリックし、インストールを終了します。

PC が再起動されると、先ほど指定したフォルダ名が以下のアイテムと共にスタートメニューに表示されます。



- **ASCET Uninstall**

  アンインストールルーチンを起動します（2.4 項を参照してください）。
- **ASCET V5.2**

  ASCET プログラムを起動します。
- **AS Editor**

  AS エディタを起動します（『ASCET ユーザーズガイド』の 4.14 項を参照してください）。
- **ETAS Network settings**

  ETAS ネットワークの設定を行うツールを起動します。
- **LicenseInfo**

  “Obtain License Info”ダイアログボックスを開きます（第 3 章を参照してください）。
- **Online manuals**

  オンラインマニュアルがインストールされている場合は、ここからマニュアルディレクトリ ETAS¥ETASManuals を開くことができます。各マニュアルはこのディレクトリ下のサブディレクトリに保存されています。
- **PC RemoteControl**

  リモートインターフェースの設定を有効にします。
- **ReadMe**

  ASCET V5.2 についての最新情報が収められているファイルを開きます。

ASCET-MD

基本システムをインストールすると、ASCET-MD のインストールが可能になります。

ASCET-MD をインストールする：

- スタートメニューから、**ファイル名を指定して実行**を選択します。

- コマンド行に、インスタレーションファイルのパスを入力します（たとえば、CD からインストールする場合には、`d:¥ASCET5.2¥ASCET.exe` と入力します）。
- **OK** をクリックして確定します。

  インスタレーションプログラムが起動されます。すでに ASCET 基本システムがインストールされているので、パスや機能範囲などの設定作業は必要ありません。
- **Next** ボタンで次のダイアログボックスに進みます。
- 前のダイアログボックスに戻るには、**Back** ボタンをクリックします。
- インストールを中止する場合は、**Cancel** ボタンをクリックします。

2.2.2 特殊な条件でのインストール

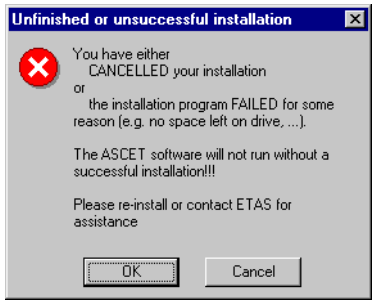
インストールを中止する：

インストールの途中で、インストールを中止することができます。以下のように行ってください。

- 操作中のダイアログボックスで、**Cancel** ボタンをクリックします。

- 元のダイアログに戻るには、**Resume** をクリックします。

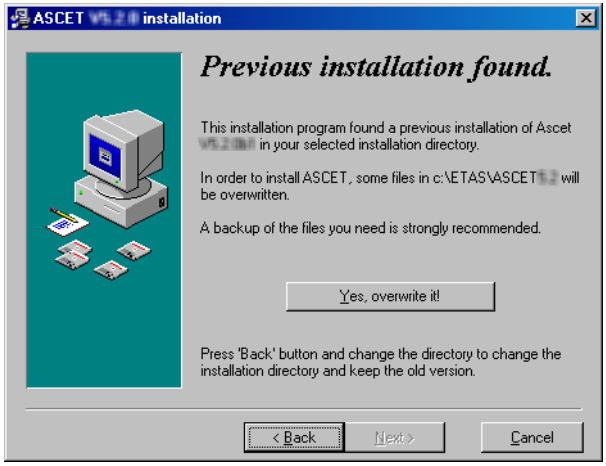
- セットアッププログラムを終了してインストールを中止するには、**Exit Setup** ボタンをクリックします。



- **OK** をクリックすると、インストールが中止されます。

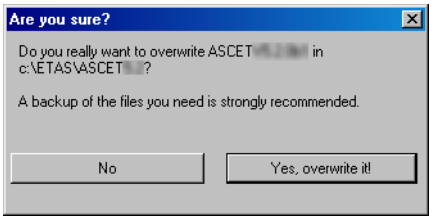
既存のプログラムバージョンに上書きする：

インストールしようとするソフトウェアの旧バージョンがすでにインストールされている場合、または指定されたディレクトリに別のソフトウェアがすでにインストールされている場合、その旨を通知するダイアログボックスが開きます。たとえば、バージョン 5.2.0 のベータバージョンがインストールされている PC にバージョン 5.2. をインストールしようとすると、以下のようなダイアログボックスが開きます。



- 表示されたメッセージをよく読んでください。

  この例では、指定のディレクトリに旧バージョンがすでにインストールされていることが示されています。以降の処理を続行すると、これらのファイルは上書きされます。
- 他のディレクトリにインストールするには、**Back** ボタンをクリックします。
- 既存のファイルに上書きしたい場合は、**Yes, overwrite it** ボタンをクリックします。

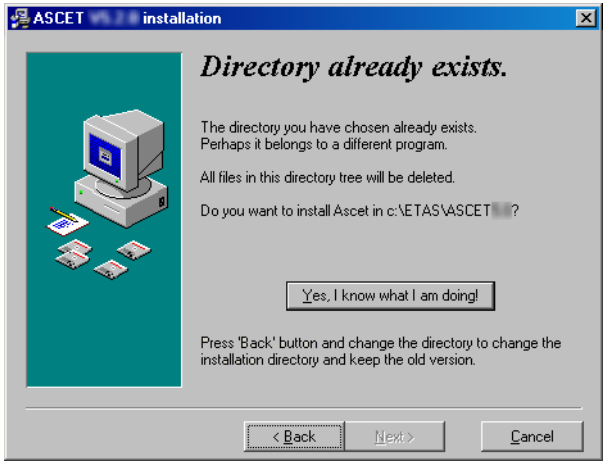


- 上記の確認メッセージが表示されるので、**Yes, overwrite it!** をクリックして確定します。

  上書きせずに元のダイアログボックスに戻るには **No** をクリックします。

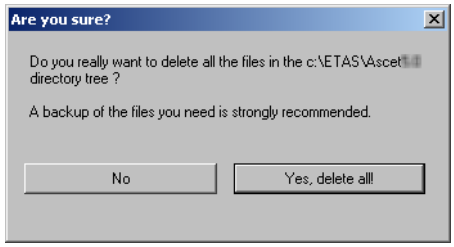
既存のディレクトリに上書きする：

指定されたディレクトリがすでに存在していて、そこに ASCET が完全にインストールされていない場合、以下のダイアログボックスが開きます。このような状況は、前回のインストール処理が途中でキャンセルされた場合などに発生します。



- 表示されたメッセージをよく読んでください。

  この例では、指定されたディレクトリがすでに存在していることを示しています。
- 他のディレクトリにインストールするには、**Back** ボタンをクリックします。
- 既存のディレクトリに上書きしたい場合は、**Yes, overwrite it** ボタンをクリックします。



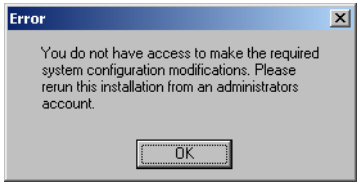
- 確認メッセージが表示されるので、**Yes, delete all!** をクリックして確定します。

  ASCET がインストールされて、既存のファイルはすべて削除されます。

  上書きを行わずに前のダイアログボックスに戻るには **No** をクリックします。

管理者権限なしでインストールを実行しようとした場合：

Windows にログオンした時のユーザー名に管理者権限が与えられていない場合、以下のメッセージボックスが開きます。ASCET のインストールを行うには管理者権限が必要であるため、メッセージを確認した後は、インストール処理は中断されます。



- メッセージを確認して **OK** をクリックします。

  インストールは中断されます。
- システム管理者の方に問い合わせてください。
- 必要な権限を取得した後、再度インストールを行ってください。

2.3 ネットワークインスタレーション

CD からだけでなく、PC 上のネットワークドライブからも ASCET をインストールできます。

ネットワークインスタレーションには、PC に実際にインストールする前にあらかじめインストールオプションを設定しておくことができるという利点があります（2.3.2 を参照してください）。

2.3.1 ファイルの準備

ネットワークドライブからインストールを可能にするため、まずインストールに必要なファイルを CD からネットワークドライブ上にコピーしておく必要があります。

ファイルをネットワークサーバにコピーする：

- 任意のネットワークドライブ上にソースディレクトリを作成します。
- CD の全データをソースディレクトリにコピーします。

インスタレーションログ

ユーザーがネットワークインスタレーションを行った履歴は、ネットワーク上のログファイルに記録されます。したがって、すべてのユーザーに、`x:¥user` ディレクトリ、または `install.ini` に指定されたディレクトリに対する書き込みアクセス権が必要です。

2.3.2 ネットワークインスタレーションのカスタマイズ

ユーザーが自分のワークステーションに ASCET をインストールする前に、ASCET のデフォルトのインストールオプションをカスタマイズしておくことができます。

ネットワークインスタレーションの場合、以下のことが可能となります。

- ディレクトリなどのデフォルト設定を変更しておくことにより、インスタレーションダイアログをカスタマイズできます。
- ASCET のインストールを、ユーザーの介入なしにバックグラウンドで完全に自動実行できます。
- カスタマイズしたファイルで製品のデータディレクトリ（デフォルト設定は `[drive]:¥ETASdata¥ASCET5.2¥...`）内のファイルを上書きしたり、既存のディレクトリにファイルが追加されるように指定できます。

インスタレーションダイアログのカスタマイズ

大規模な部署内で多くのユーザーがネットワークインスタレーションを行うような場合、インストールに使用されるオプション情報のデフォルト設定をあらかじめカスタマイズしておいて、各 PC に同じ条件でインストールが行われるように

する必要が生じる場合があります。このようなカスタマイズは、install.ini コンフィギュレーションファイルを使えば可能です。このファイルは、インスタレーションディレクトリに格納されています。

以下の例に従って、デフォルト設定を変更してください。

コンフィギュレーションファイルをカスタマイズする：

- install.ini ファイルをテキストエディタで開きます。

  以下は、この INI ファイルのデータの一例です。

```
;Sets the main directory of ASCET
;MainDir=c:¥etas¥ASCET5.2
```

- デフォルト設定を修正するには、MainDir キーワードの行の“;”（コメントを示す）を削除します。
- パスを、たとえば H:¥programs¥etas¥ASCET5.2 というように変更します。

  ファイルの内容は、以下のようになります。

```
;Sets the main directory of ASCET
MainDir=H:¥programs¥etas¥ASCET5.2
```

- install.ini の他の部分も、同じ方法で適宜変更します。
- 変更内容を保存してから、エディタを閉じます。

これで、ASCET.exe を実行してインストールを開始すると、ダイアログボックスには新しい設定がデフォルトオプションとして表示されるようになります。

注記

パス設定の変更により、以下のようなカスタムインストールが可能です。

自動インストール

ASCET.exe ¥s というコマンドを実行すると、ASCET のインストールがバックグラウンドで完全に自動実行されます。ユーザーの介入は一切必要なく、その時点で有効なデフォルト設定が自動的に選択されます。install.ini ファイル内の各デフォルト設定は任意に変更できます（31 ページの「インスタレーションダイアログのカスタマイズ」を参照してください）。

システム管理者が“ASCET.exe ¥s”コマンドを含むバッチファイルを作成し、install.ini 内の必要な設定を行えば、ユーザーはこのバッチファイルを実行するだけで、一切の入力作業なしにインストール作業を実行できるようになります。

なおこのインストール方法においてはダイアログボックスはまったく表示されないので、インストールが完了した時点でユーザーに通知を行うメカニズムを用意しておくとよいでしょう。

ASCET ファイルのカスタマイズ

以下に説明するようなカスタマイズ機能を利用してインストールプログラムを調整し、インストール実行時にカスタマイズされたファイルをデフォルトファイルに上書きしたり、他のファイルをインスタレーションセットに含めたりすることができます。

この機能によって、カスタマイズされたデータベース、ユーザープロファイル、およびダイアログボックステンプレートをインストールされるプログラムに統合することが可能となります。

このしくみは比較的単純です。インスタレーションディレクトリ下に `InstData¥...` というサブディレクトリを作成し、正しいディレクトリ構造を維持しながら、カスタムファイルをそこにコピーします。

カスタムファイルを作成するには、まず ASCET をテスト用 PC にインストールし、これを用いてファイルを作成します。

ASCET のデフォルトインストールを行うと、`ETASData¥ASCET5.2¥...` ディレクトリに以下のようなサブディレクトリが作成されます。そこには ASCET のデフォルト設定を定義するファイルがあり、それらをカスタマイズすることが可能です。

- `Database¥DB¥`

  `db` サブディレクトリには、デフォルトのデータベースが格納されています。ここに、たとえば `Database¥DemoDB¥` といった別のデモ用データベースを作成できます。

- `User¥[user name]` - Windows のログインユーザー名により異なる

  `[user name]` サブディレクトリには、デフォルトのユーザープロファイルが格納されます。設定可能なオプションはすべて、このサブディレクトリに保存されます。

ネットワークインスタレーション用のデータをカスタマイズする：

- ASCET を PC にインストールします。
- ASCET を起動します。
- ユーザープロファイルを修正します。
- データベースを修正するか、新しいデータベースを追加します。
- ASCET を終了します。

ここまででカスタマイズは終了したので、これらのファイルをインスタレーションプログラムに統合します。次の 2 通りの方法があります。

- 同じ名前の既存のファイルにカスタムファイルを上書きします。これを行うためには、InstData¥overwrite¥ というフォルダをインスタレーションディレクトリに作成しておく必要があります。
- カスタムファイルの名前を変更して、それらを既存のファイルに追加します。同じ名前のファイルは上書きされません。これを行うためには、InstData¥add-only¥というフォルダをインスタレーションディレクトリに作成しておく必要があります。

カスタムファイルをインスタレーションプログラムに含める場合、必ずその親ディレクトリもコピーしてください。ETASData¥ASCET5.2¥ ディレクトリのレベルは、InstData¥overwrite¥ や InstData¥add-only¥ と同じでなければなりません。

例：

```
InstData¥overwrite¥user¥userDef.ini
InstData¥add-only¥database¥additionalDB¥
```

修正済みのユーザープロファイルを統合する：

- カスタマイズした ETASData¥ASCET5.2¥user¥[user name]¥ascetsd.ini ファイルを InstData¥overwrite¥user¥ サブディレクトリにコピーします。
- ascetsd.ini ファイルの名前を userDef.ini に変更します。

  これで、インストール後は、新規のユーザー用としてこの初期化ファイルが使用されるようになります。

修正を加えたデータベースを統合する：

- カスタマイズしたデータベース、つまり ¥database¥DB¥ サブディレクトリを InstData¥add-only¥... サブディレクトリにコピーします。
- DB ディレクトリの名前を任意に変更します。変更しないとデータベースはコピーされません。

  もちろん、InstData¥overwrite¥ ディレクトリを用いて DB データベースを上書きすることもできます。

ASCET.exe でインストールルーチンを起動すると、デフォルトファイルはカスタムファイルで上書きされ、対応するディレクトリに新しいファイルが追加されます。

2.3.3 ネットワークドライブからの ASCET のインストール

ネットワークドライブからのインストールは、CD-ROM から行う場合と同じ手順でインストールできます。ネットワークドライブ上のどのディレクトリにインストールプログラムが格納されているかを確認してから、19 ページの「インストール」の項を参照してインストールしてください。

注記

ネットワークドライブから ASCET をインストールするには、ネットワークドライブのログディレクトリへの書き込みアクセス権が必要です（2.3.1 を参照してください）。

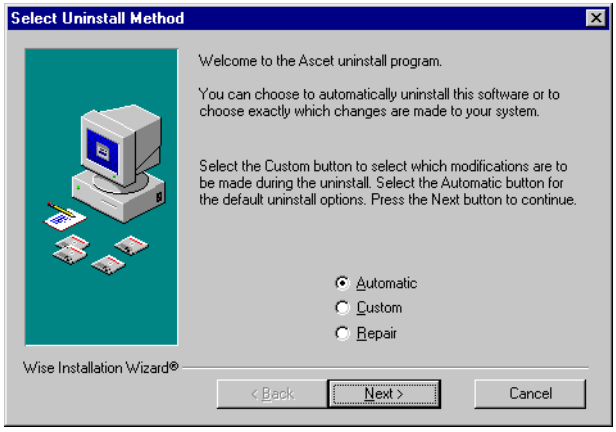
2.4 ASCET のアンインストール

ASCET をアンインストールすると、インストールされている ASCET 製品すべてが自動的にアンインストールされます。各 ASCET ファミリ製品（ASCET-MD、ASCET-SE、ASCET-RP）やアドオン製品を個別にアンインストールすることはできません。

2.4.1 自動アンインストール

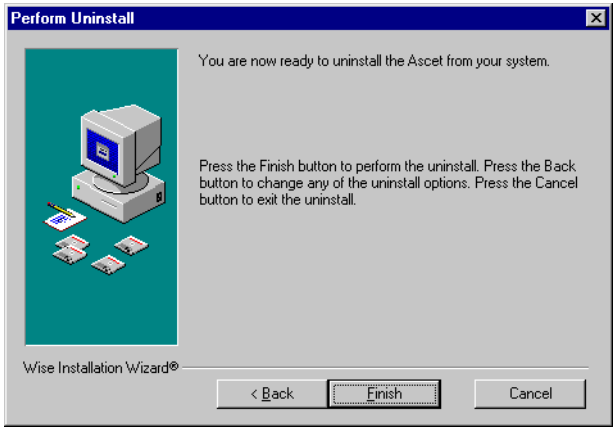
ASCET を自動アンインストールする：

- スタートメニューのプログラムグループから、**ASCET Uninstall** を選択します。
  以下のダイアログボックスが開きます。

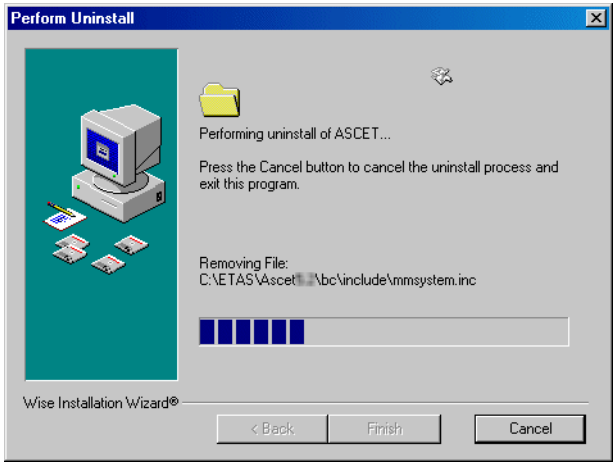


- **Automatic** を選択します。

- **Next** ボタンをクリックします。



- アンインストールを実行するには **Finish** ボタンをクリックします。



アンインストール実行中にアンインストール処理を中止することもできます。
Cancel ボタンをクリックすると、以下のダイアログボックスが開きます。

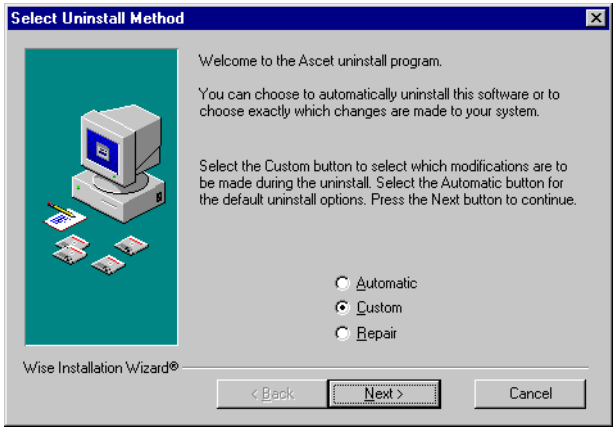
注記

キャンセルした時点でデータがすでに消去されていた場合は、ASCET を再インストールする必要があります。

2.4.2 カスタムアンインストール

ASCET をマニュアル操作でアンインストールする：

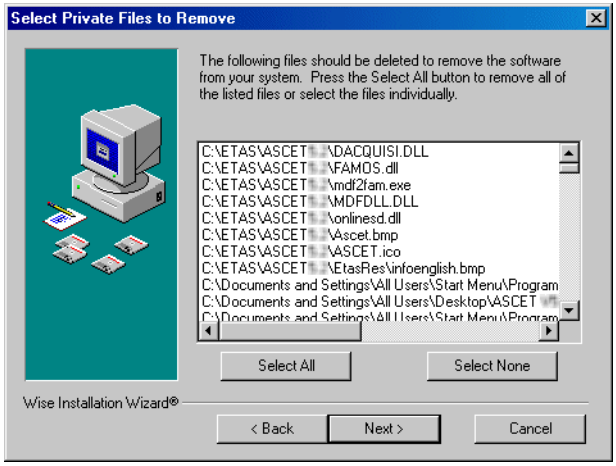
- スタートメニューのプログラムグループから、**ASCET Uninstall** を選択します。
  以下のダイアログボックスが開きます。



- **Custom** を選択します。

- **Next** ボタンをクリックします。

  “Select Private Files to Remove”ダイアログボックスが開きます。



- “Select Private Files to Remove”ダイアログボックスで、削除したいファイルを選択します。
- **Next** ボタンをクリックします。
- “Select Directories to Remove”ダイアログボックスで、削除したいディレクトリを選択します。
- **Next** ボタンをクリックします。
- “Select INI Files to Remove”ダイアログボックスで、削除したい `*.ini` ファイルを選択します。
- **Next** ボタンをクリックします。
- “Select INI Items to Edit”ダイアログボックスで、編集したい `*.ini` エントリを選択します。
- **Next** ボタンをクリックします。
- “Select Registry Keys to Remove”ダイアログボックスで、削除したいレジストリキーを選択します。
- **Next** ボタンをクリックします。
- “Select Registry Trees to Remove”ダイアログボックスで、削除したいレジストリフォルダを選択します。
- **Next** ボタンをクリックします。

- “Select Registry Keys to Edit”ダイアログボックスで、編集したいレジストリキーを選択します。
- **Next** ボタンをクリックします。
- “Select Sub-Systems to Remove”ダイアログボックスで、削除したいサブシステムを選択します。
- **Next** ボタンをクリックします。
- “Perform Uninstall”ダイアログボックスで、**Finish** ボタンをクリックします。

  アンインストールが実行されます。

カスタムアンインストール実行中でも、自動アンインストールの場合と同様に **Cancel** ボタンでアンインストールを中止することができます。

注記

キャンセルした時点でデータがすでに消去されていた場合は、ASCET を再インストールする必要があります。

3 ライセンスについて

ASCET 製品ファミリに含まれるすべての ASCET 製品とそのアドオン製品は、ライセンス管理の対象となります。製品をインストールした後に ASCET を使用するには、ご使用の PC 専用のライセンスファイルを ETAS から入手し、それを PC にインストールする必要があります。このファイルがない場合、ASCET をインストールすることはできますが、実際に使用することはできません。

3.1 ライセンスの取得

ライセンスファイルの取得とインストールは、以下のように行ってください。

ライセンスファイルを取得する：

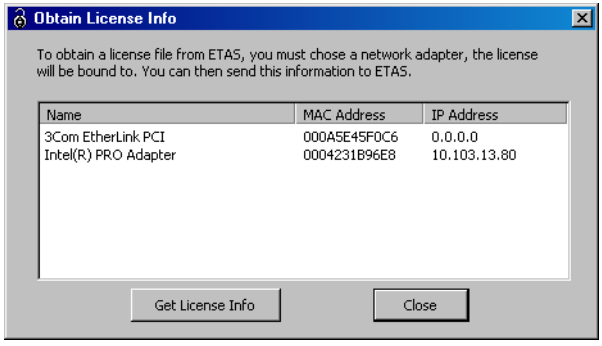
ライセンスは、特定のユーザー名と PC に対して割り当てられます。ライセンスファイルを作成するために必要な情報を得るには、ASCET プログラムグループ内の **LicenseInfo** ツールを使用します。

注記

ASCET をインストールする前にライセンスファイルを ETAS から入手するには、製品 CD-ROM 内の *LicenseInfo.exe* を起動してください。

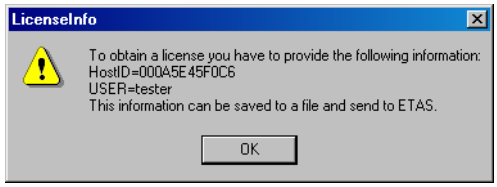
- Windows のスタートメニューから、ご使用のバージョンの ASCET フォルダを選択します。
- そのフォルダ内の **LicenseInfo** を選択します。

  “Obtain License Info”ダイアログボックスが開きます。PC に組み込まれている各ネットワークアダプタ（ネットワークカード）のホスト ID（MAC アドレス）と IP アドレスが表示されます。



- リストアップされているネットワークアダプタのいずれかを選択し、**Get License Info** をクリックします。

  その PC 用のライセンスファイルを作成するために必要な情報が収集され、“License Info”ダイアログボックスに表示されます。



- 表示された情報を所定のテキストファイルに保存する必要があるので、**OK** をクリックします。
- ファイル選択ダイアログボックスで、テキストファイルのパスとファイル名を設定します。
- **保存** をクリックします。

  ファイルが作成され、テキストエディタで自動的にそのファイルが開きます。このファイルには、ユーザー名とホスト ID のほか、ユーザーの E メールアドレス、ライセンス番号、その他の情報を入力する行と、ファイルの送信先アドレスが含まれています。

  ライセンス契約書に記載されたライセンス番号を入力し、必要に応じてその他の情報を入力し、最後にそのファイルを保存してください。
- “Obtain License Info”ダイアログボックスを閉じます。

- 完成したテキストファイルを E メールに添付して ETAS に送信します。

  ASCET の場合、各 ASCET 製品（ASCET-MD、ASCET-MIP など）ごとにファイルを作成してお送りいただく必要があります。

  ETAS から、ユーザー ID とホスト ID に対応するライセンスキーが格納されたライセンスファイルをお送りします。

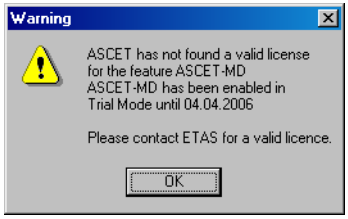
注記

ETAS から送られてきたライセンスファイルは**編集しないでください**。編集されたライセンスファイルは無効になってしまいます。

- ライセンスファイルを、インストール時に指定したディレクトリに格納します（25 ページ「ライセンスファイルの格納場所を指定する：」を参照してください）。
  ライセンスファイルディレクトリはデフォルトでは `ETAS¥ETASShared3¥Licenses` ですが、インストール後、`ETAS_LICENSE_FILE` という環境変数の値を変更することにより他のディレクトリに変更することもできます。

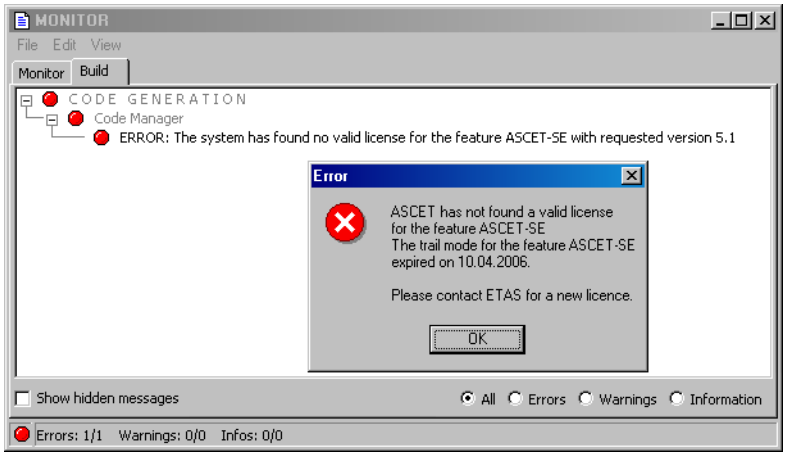
  プログラムを起動すると、このライセンスディレクトリにあるライセンスファイルが自動認識されます。

インストール時に指定されたディレクトリ（25 ページ参照）内に有効なライセンスファイルが見つからなかった場合、ASCET-MD、ASCET-RP、ASCET-SE はトライアルモードで起動します。この場合、ある一定の期間は通常どおりに操作を行えますが、ライセンスファイル検索が定期的に行われ、ファイルが見つからない場合はワーニングメッセージが表示されます。



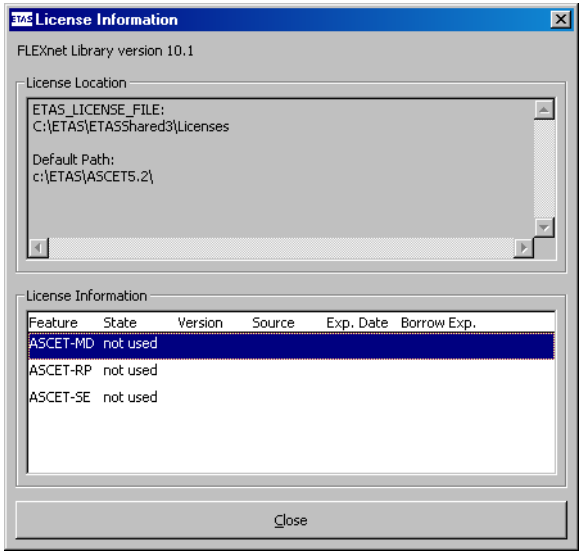
アドオン製品（ASCET-MIP、INTECRIO-ASC など）にはトライアルモードはないので、ライセンスなしには操作を行えません。

ライセンスファイルがない状態で所定のトライアル期間が経過すると、ワーニングメッセージの代わりにエラーメッセージが表示され、ライセンスファイルが見つかるまでツールは使用できなくなります。



3.2 ライセンスステータス

メニューコマンド **Help → License Info** を選択すると”License Information”ダイアログボックスが開き、現在のライセンスの状態が表示されます。



このダイアログボックスの上部には、ライセンスファイルの格納場所が表示され、下部の "License Information" テーブルには、現在のライセンスステータスが各機能ごとに一覧表示されます。

カラム	説明
`Feature`	機能（インストール済みの ASCET 製品およびアドオン製品）
`State`	ライセンスステータス（以下のいずれかの値です） `not used` – ライセンス未要求（ツールの起動後、機能が使用されていない） `licensed` – 有効なライセンスで運用中 `grace mode` – トライアルモードで運用中 `unlicensed` – トライアルモード期間終了済み
`Version`	ライセンスファイルに基づくバージョン番号
`Source`	ライセンスファイルのソース（以下のいずれかの値です） `local license` または サーバー名（ライセンスがない場合は空欄）
`Exp. Date`	ライセンスまたはトライアルモードの終了日付
`Borrow Exp.`	ライセンス借用期間の終了日付（3.3 項参照）

表 3-1　"License Information" テーブルの内容

3.3　ライセンスの借用

サーバーライセンスを使用している場合、一時的にローカルライセンスが必要になった際に（ラップトップ PC で車上テストを行うような場合）、限られた期間だけサーバーからライセンスを「借用」することができます。サーバーに接続した状態で作業を行う場合は、ライセンスの借用は必要ありません。

デフォルトの借用期間は 60 日ですが、これは ASCET オプションダイアログボックスの "General settings" タブで変更できます。

デフォルトの借用期間を変更する：

- 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。
- "Lincense" ノードを開きます。
- "Borrowing license for days" フィールドに借用期間（日数）を入力します。

- "Options”ダイアログボックスを閉じます。

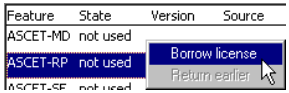
  新しい借用期限は、次にライセンスを借用する時から有効になります。

  現在借用しているライセンスの期間は、変更されません。

ライセンスを借用する：

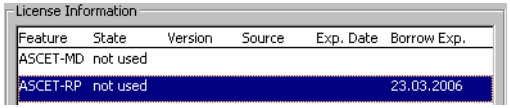
- コンポーネントマネージャでメニューコマンド **Help → License Info** を選択し、”License Information”ダイアログボックスを開きます。

  ライセンスの借用は、`not used` ステートのすべての機能について行えます。



- "License Information”ダイアログボックスで、ライセンスを借用したい機能を右クリックしてショートカットメニューを開き、**Borrow license** コマンドを選択します。

  これによりライセンスの借用が行われ、借用したライセンスの有効期限が”License Information”テーブルに表示されます。



  選択された機能が、PC 上でフルに操作できるようになります。

借ライセンスが借用されている期間は、他のユーザーがそのライセンスを使用することができません。そのため、ライセンスを他のユーザーが使用できるように、借用期限前にライセンスを返却することが必要となる場合があります。

ライセンスを借用期限前に返却する際は、PC がネットワークに接続された状態で以下の操作を行ってください。

借用ライセンスを返却する（通常時）：

- "License Information”ダイアログボックスを開きます。
- ライセンスを返却したい機能を選択してショートカットメニューを開き、**Return earlier** コマンドを選択します。

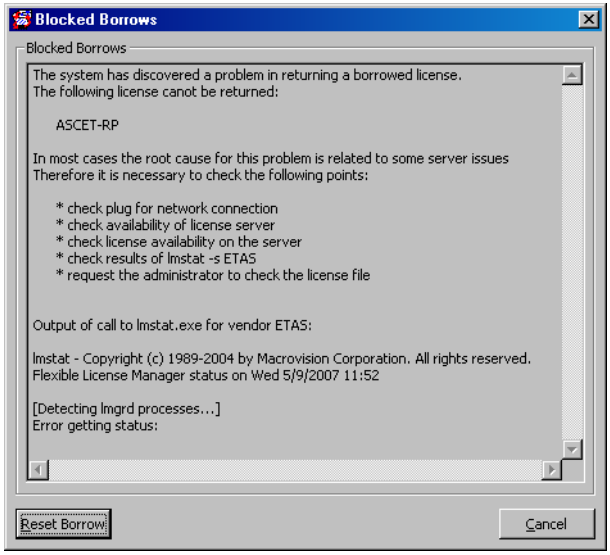
  ライセンスがサーバーに戻ります。

サーバーライセンスの借用中は、借用に関する情報がサーバーと PC の両方に保存されていますが、サーバー側に保存された情報が何らかの原因（サーバーの再起動の失敗やその他の障害など）によって失われた場合、上記の **Return earlier** コマンドによる返却処理は行えなくなります。このような場合は、同コマンドを使用して PC 上のローカル情報を削除してください。

借用ライセンスを返却する（障害時）：

- "License Information" ダイアログボックスを開きます。
- ライセンスを返却したい機能を選択してショートカットメニューを開き、**Return earlier** コマンドを選択します。

  "Blocked Borrows" ダイアログボックスが開き、チェックの必要があるポイントが表示されます。



- 以下のポイントをチェックしてください。
  - PC がネットワークに接続されているか
  - ライセンスサーバーが正常に起動し、ネットワークに接続されているか
  - サーバー上にライセンスが存在しているか
  - コマンドラインから `lmstat -s ETAS` を実行し、その結果を確認
  - サーバー上にライセンス情報ファイルが存在しているか（管理者の方にお尋ねください）

- 上記のチェックによってサーバー上にライセンス情報が存在していないことが確認された場合に限り、**Reset Borrow** をクリックしてください。

注記

サーバー上にライセンス情報が存在している状態においのローカルライセンス情報を削除してしまうと、設定済みの借用期限が経過するまで、このライセンスは**使用不可能**になります。

PC 上に保存されている借用ライセンスについての情報が削除されます。

4 ASCET を理解する

ASCET は、既存のコンポーネントを正確に再利用しながら非常に高品質な組込みソフトウェアを構築することを可能にする、先進的な開発ツールです。独自のグラフィック開発環境によりターゲットに依存しない機能記述を行うことができ、ブロックダイアグラムで記述された制御機能から組込みシステム用の制御ソフトウェアを自動生成できます。また運用時のターゲットシステムと同じリアルタイム環境でプロトタイピングを行えるので、開発サイクルの早い段階でのテストの実施が可能です。

ASCET を含む ETAS のツールチェーンを使用すれば、ツールに対する投資をフルに活用でき、確実に収益を高めることが可能です。

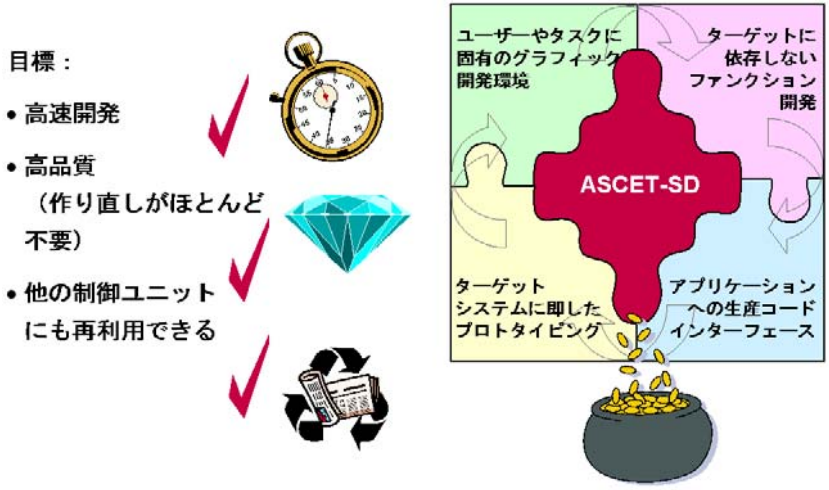


図 4-1 ASCET 開発環境のメリット

4.1 ECU 開発の効率向上

過去 20 年間で、ECU（電子制御ユニット）は自動車業界に徐々に浸透し、近年では、機能性、速度、ネットワーキング機能を求める声が急激に高まりました。これらに関する技術や手法は大きく変化し、今日、エレクトロニクスは新しい車両設計の成功を左右する主要因になっています。そして開発のコストと速度がますます重要視されている今日では最新技術の導入が不可欠であり、ETAS はその分野で常に新たな技術を提供します。

4.1.1 現代の組込み制御システムの技術的役割

今日の ECU の特徴の 1 つに、多くの高度な制御ファンクションが 1 つのプロセッサチップに集積されている点があげられます。実際に運用で使われるマイクロコントローラは、複数の I/O チャンネルと通信チャンネルを装備し、外のさまざまな事象がプロセッサに直接つながっています。

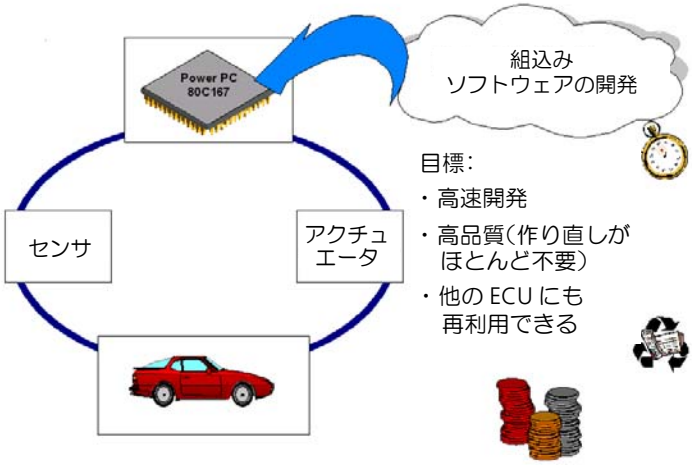


図 4-2 組込み制御システムの基本概念

1 台の車両を制御する組込み制御システムには、複雑な制御回路がいくつも実装されています。制御対象となるシステムの現在の状態についての情報は、センサにより正確に読み取られて過去のデータと共に処理され、その結果として、対応するアクチュエータへ制御情報が出力されます。この一連の流れにおいてはシステムの各種要素をスムーズに結合することが重要です。各種 ECU をネットワークで結び付けることによってシステム全体の管理が可能となり、安全性や、さらに排出汚染物質や燃費の低減、といった現在と未来の環境のために必要な条件を実現できるようになります。

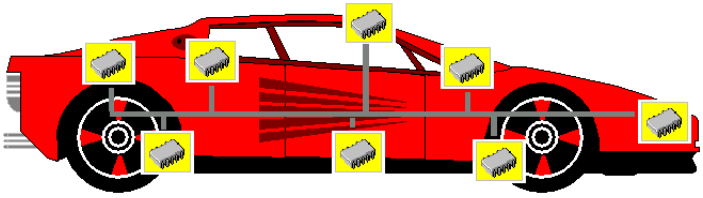


図 4-3 組込み制御の応用

自動車用エレクトロニクスは、すでに車両システムのあらゆる部位に浸透しています。歴史的に見ると、エンジンとトランスミッションの制御から始まり、今ではブレーキやサスペンションなどの制御においても ECU が使われています。またさらに、エアバッグ、コックピット、ナビゲーションシステムなどにもエレクトロニクスは幅広く浸透しています。

組込み制御システム

組込み制御システムを設計する場合、詳細にわたるさまざまな検討が必要です。通常、ECU は数多くの外部信号を取り込む必要があり、しかも、各信号間の物理的な関係は非常に複雑です。設計エンジニアが迅速に結果を得ようとすれば、マイクロプロセッサに ECU の機能を実装する際のさまざまな難題を解決してくれるメソッドとツールが不可欠です。必要な機能を抽象化し。物理レベルで仕様検討を可能にすることにより、複雑さを軽減し、設計をより単純なものにすることができなければなりません。

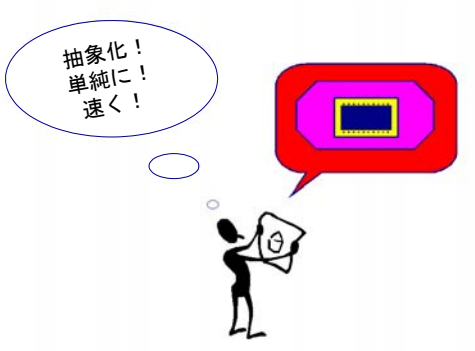


図 4-4　組込み制御の構想

このように、組込み制御の開発システムに求められる抽象性、単純さ、高速性は、すべて ASCET で実現されます。ASCET を使用すれば、設計エンジニアは組込み制御システムの物理的特性に十分に集中でき、抽象的かつ単純な環境で ECU の機能を設計し、最終的に量産 ECU 用コードを生成することができます。これにより、きわめて迅速に設計を遂行できます。

近年、ソフトウェア開発においては、どの分野においても抽象性、単純さ、高速性が求められています。その結果、設計とその分析を視覚的に行う方法が考案されました。このためのさまざまな技術が UML（**U**niversal **M**odeling **L**anguage：汎用モデリング言語）という汎用言語に集結されています。「パターン」として知られる概念は、このレベルで定義されます。パターンは、設計エンジニアの経験をかつてないほど抽象的な形で表すものです。しかしこのアプローチでは、リアルタイム動作についての記述はサポートされていないため、そのまま実際の制御に適用することはできません。それに対して ASCET は、適度の抽象化、リアルタイム要件の明解なサポート、制御ブロックダイアグラムとステートマシンによる視

覚的記述、といった手法でこの問題点を解決しています。機能記述を UML から ASCET に転送したり、逆に戻すこともでき、ASCET のこの**オープンな特性**が新しい可能性を切り開きます。

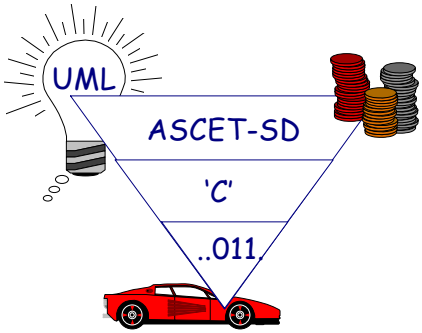


図 4-5　最新のソフトウェア開発

ASCET では、ボタン操作 1 つで機能記述を C コードに変換できます。このコードはさらに、ターゲット専用ツール（コンパイラ、リンカ、デバッガ）により、実行形式のバイナリコードに変換され、ターゲットにロードされます。ASCET は開発時間を短縮するので、コストも低減できます。しかも、純粋な C コードで開発された成果物を ASCET に移行する際のサポート機能も優れているので、既存のソースコードの有効利用が可能です。またリアルタイム要件の抽象化は、ETAS の **OSEK** 準拠のオペレーティングシステム ERCOS^{EK} により完全にサポートされています。

だたし、実行形式のコードを生成しても、組込み制御システムの開発が終了するのはまだ先の話です。テストと適合の段階では、広範にわたる測定が行われ、パラメータが最適化されます。この段階で開発システムに要求されるものは、適合を行うためのツールです。ここでは、連続性のある開発工程を実現するため、各種インターフェースやフォーマットのサポートが要求されます。ETAS は ASCET、LABCAR、INCA、といったツールチェーンにより、これらの要求に総合的に対応します。**標準化されたインターフェース**をサポートする製品群により設計上の便宜が図られるので、設計エンジニアは目下の主要業務である組込み制御システム

の設計に集中することができます。またドキュメント作成を設計と並行して連続的に進められるので、プロジェクト管理（コンフィギュレーション管理）を効率的に行うことができます。

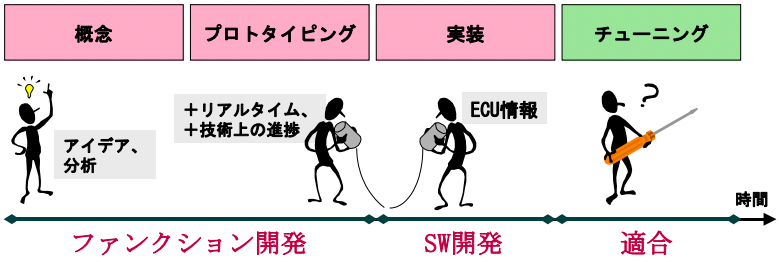


図 4-6　連続性のある工程 - オープンインターフェース

各工程に連続性がないと、工程間のデータ変換に時間を取られることになります。これにより、仕様についての誤った理解やずれ、あるいはエラーさえ発生する恐れもあります。しかも、この問題は仕様に変更が加えられるたびに生じる可能性があり、そのために膨大なコストがかかってしまいます。このような事態は、連続性のあるツールチェーンを選択することによって回避でき、コストも大きく低減できます。

自動車業界に焦点を絞って

ASCET は自動車制御に特化して開発されました。その結果、他の分野ではまだ明確になっていなかった多くの必要条件が明らかになり、さらに、他の工程でも利用できる多くの一般概念が EATS 社のツールチェーンに採り入れられました。

自動車組込み制御システムの特徴は、マップなどによる単純な制御概念が求められていることです。ここではメモリサイズとパフォーマンス（処理速度）の制約が非常に重要になります。理想的なコントローラを単純化したモデルによってのみ、実際のプロセスを監視し制御するために必要な結果を得ることができます。さらに、大量に生産される量産 ECU について考えると、組込み制御システムの品質は非常に重要な要素になります。そしてもう 1 つの要素は安全性です。自動車に実装されている多くの ECU は、安全性に関する動作や安全性が非常に重視される動作に影響を与えます。ブレーキ制御やエンジン制御、さらにはパワーウィンドウ（手足を挟む危険性があるため）などが、特徴的な例です。このような安全性の実現には、ネットワーキングという概念が必要となります。

一方、開発用のハードウェアもさまざまな条件に対応している必要があります。シミュレーションシステム、測定／適合システム、テストシステムは、夏季や冬季の厳しい気候条件に耐えられなければなりません。また優れた電磁環境適合性を備えていなければならず、同時に最高のパフォーマンスに対する要求を満たす必要があります。

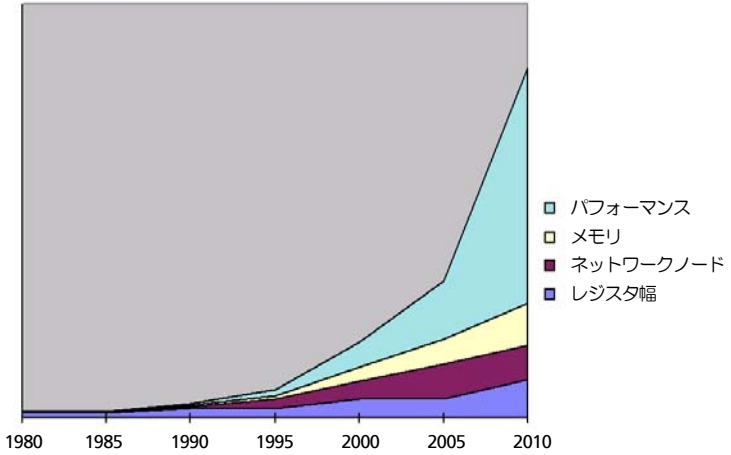


図 4-7　自動車制御に要求されるシステム要件

自動車制御システムの開発サイクルは、主として試験に要する期間によって決まります。製品の寿命は開発サイクルよりもはるかに長く、この期間中にモデルチェンジなども行われます。したがって、マイクロプロセッサ技術の強化は長期的視野で考えられるべきであり、プロセッサの変更が必要な場合（たとえばメモリ技術の向上などにより）は、開発作業に影響を与えずにそれを行えるようにしなければなりません。そこで最初は、レジスタ幅（プロセッサ性能の尺度）は 5 年に 1 度だけ倍増させるというような、比較的大まかなモデルから始めます。この場合、大幅なオーバラップの発生を想定する必要があります。つまり、初の 32 ビットプロセッサが導入された時点では、多くのプロジェクトは 16 ビットプロセッサを使用しており、中には依然として 8 ビットのものを使用しているプロジェクトもありました。今後、これと同じことが、整数一辺倒の演算から浮動小数点演算への革新の際にも想定されます。

4.1.2　開発工程におけるコスト面での課題

開発の方法や手順は各会社によって異なりますが、一般的に求められる共通の要件も多くあり、それらは最終的に支援ツールに反映されています。

今日の開発作業においては、時間、コスト、品質および柔軟性の 4 点が、戦略上決定的な役割を担っています。この 4 つの側面から開発工程を評価する必要があります。連続性のあるツールチェーンの導入は、この方向で工程を改良するためにはきわめて重大なステップです。ただし、チェーンのすべてのメンバーが開発に十分に寄与できなければ、成功は望めません。ETAS の ASCET やその他のツー

ルは、この難問を解決します。これらのツールを使用すれば、標準化されたインターフェースや、組込み制御システムの技術要件に関する共通の手法により、開発時間の短縮や、コストの低減が実現できます。また、制御関係のモデリングが的確に行われる、という点でもコストを抑えることができ、このことが、早い段階でのプロトタイピングを可能にすると共に、不必要な反復と不良部分の発生を防ぎ、開発結果の品質も向上させます。ASCET のコード自動生成機能により、通常起こりうる実装時の誤差がなくなります。最終製品であるソフトウェアの挙動が仕様を 100% 満たすようになり、面倒な手直しも不要になります。また、独自に設計された ASCET データベースにより、開発作業は非常に融通性に富んだものになります。モジュールは任意の組み合わせでつなぎ合わせることができ、コード自動生成機能が最適化と調整を行います。そしてさらに、ASCET は容易に修正できるユーザーインターフェースを提供し、これがすべての開発工程を柔軟にサポートします。

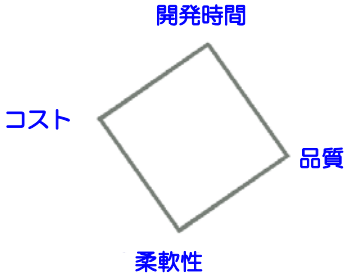


図 4-8　戦略的フォアサム - 製品開発に求められる 4 つの要因

自動車業界に特有なコスト面での課題は、生産工程に関して考えれば、いかに大量の製品を扱うか、という点が考慮されますが、開発エンジニアの立場から見ると、たとえば 1 つのプロジェクトにおける複数のサプライヤ間の協力など、他の側面での問題点があります。したがって、ツールチェーンには、チームサポートというテーマの下にまとめられる各種条件が課せられることになります。データを共同で管理したり、長距離間で確実かつ再現可能な方法で交換できなければなりません。ノウハウが外部に漏れないようにすることも重要です。ASCET はこれについて適切なコンセプトを提供し、ツールに必要なインターフェースを有しています。

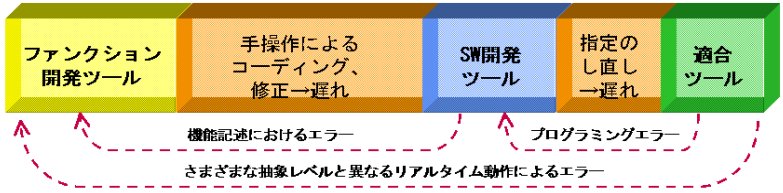


図 4-9　従来の開発工程

激しい競争が繰り広げられている市場においては、コスト削減と技術革新についての強い要求がつきものです。ここでの目標はプロジェクトの境界を越えて協力関係を達成することで、これは、モジュールの枠を越えるソリューション（同一部品コンセプト）によってのみ、実現できます。つまり、組込み制御システムのどの開発レベル（ハードウェア、オペレーティングシステム、プロトコル、ファンクション）でも、できる限り標準的なインターフェースを使うようにします。このソリューションにより、開発サイクルを大きく短縮できます。標準化は、ETAS にとってはごく普通のことです。ETAS のツールは、組込み制御システムの分野におけるスタンダード（例、ASAM、CAN、MSR、NEXUS、OSEK、VME）をサポートし、来たるべき標準化への要望に積極的に取り組んでいます。

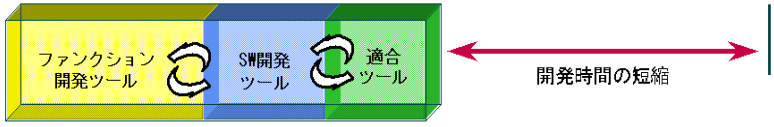


図 4-10　ASCET により最適化された開発工程

今日では、開発は効率的かつ効果的に行われて初めて経済的要求が満たされます。会社の規模とは無関係に、適切な手段をとることが基本です。毎日の作業を ASCET で行うことにより、組込み制御システムの開発効率を向上できます。これまで説明したような日々の開発作業の簡素化により、エンジニアの手間が省けるので、エンジニアは本来の作業（**新しい、革新的な ECU の考案と作成**）に集中できます。

4.1.3　革新的な技術 - 技術的展望

Embedded Control Systems（組込み制御システム）用のソフトウェアが開発されるようになってから、数十年になります。その間、使用されるプログラミング言語やツールは変わってきましたが、基本的な開発方法には変化が見られませんでした。これまで、この種の制御システムの複雑なプログラミングを行えるのは、マイクロコントローラの専門家だけでした。トラブルシューティングに用いられるデバッガの機能にも限界があり、リアルタイム処理上の問題からごく限られた範囲でしか活用できませんでした。またファンクション開発者は、最終結果を直接目にすることができませんでした。このような状況の中、ETAS がこの分野に大変革をもたらしました。ASCET の先端技術により、ドラフトの制御設計をターゲットシステム上で直接検証することが、初めて可能になったのです。

未来指向と革新的な力が ETAS の特性です。これらの特性が、進歩と改良のための推進力になっていて、ETAS の製品はこの基本概念に呼応しています。標準化はそのためのものであり、製品最適化の際には、ユーザーの要望を十分に取り入れています。技術改新を一歩先取りして、組込み制御システムの未来を ETAS と共に創り出していきませんか？

ビットコーディングの時代は終わり、組込み制御システムを実装するための言語としてアセンブラが使用されることも稀になりました。今日、多くの開発は C で行われます。この高水準言語は、ターゲットシステムに組み込むための開発形態の最終到達地点とも言えるものですが、この手段では、異なるプラットフォームでの制御のコンセプトに応用できません。もしも同じデータをそのまま再利用すれば、大変な事態になってしまいますから、ターゲットシステムが変わるたびに、同じような開発を最初から行うことになります。これは実に無駄な作業です。

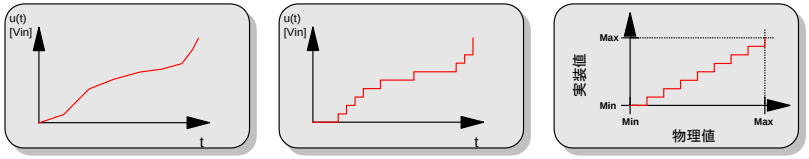


図 4-11 物理パラメータからコード実装へ

この点に関して、ASCET は無比の力を発揮します。グラフィックの制御ブロックダイアグラムからマイクロコントローラ上のソフトウェアに直接移行することが、初めて可能になりました。ASCET コードジェネレータはユニークで効率的、しかも使い方は簡単です。それだけに ETAS の設計開発者に課せられた作業は容易なものではありませんでした。まずは、必要条件を正確に抽象化する段階が困難でした。マイクロコントローラは依然として整数の算術演算を使用して機能します。また、そのメモリスペースは限られていて、ハードウェアアーキテクチャは多種多様です。そこで、各種マイクロコントローラターゲットに対応する ASCET-SE の機能により、マイクロコントローラのハードウェア上の差異を封じ込めて、標準化したインターフェースをユーザーに提供することに成功しました。その結果、1 つのマイクロコントローラから別のタイプのマイクロコントローラや最新のアップグレード版に変更しても、何も問題が生じなくなりました。もはや「再利

用」は単なる言葉上のことではなくなり、設計者の日々の作業において現実のものとなりました。ETAS はこの方法を続けていくために、新しく開発されるすべての主要プロセッサに対応する ASCET-SE を提供していきます。

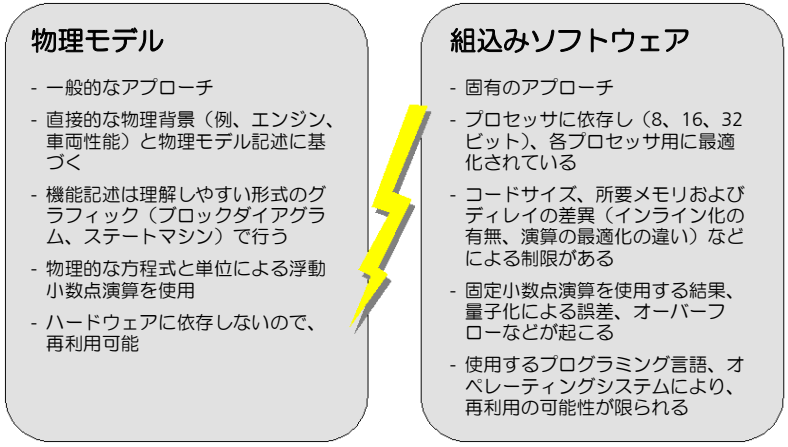


図 4-12 物理モデリングと組込みソフトウェアの相反する特徴

ASCET-SE は C コードを生成し、以降の処理には各マイクロコントローラ用の一般的な開発ツール（コンパイラ、リンカ、ロケータおよびデバッガ）を使用します。これらのツールを適切に扱うにはいくつかの問題もありましたが、ASCET は、画期的な方法でこれらを解決しました。進歩したのはこの点だけではありません。たとえば、デバッグの技術を電子制御システムの適合作業に結び付け、ターゲットシステムのリアルタイムレスポンスを完全に制御できるようになりました。ターゲットシステムの実際の制御レベルで直接作業することが、初めて可能となったわけです。そして、従来は不可欠だった、機能レベルからプロセッサビットレベルへの思考の転換が、もはや必要なくなったのです。

プロトタイピング

一方では、複雑な制御システム開発の不可欠な一要素である「シミュレーション手法」が確立されました。シミュレーションは、新しいファンクションの設計を早い段階で検証するための手法の 1 つです。

ほとんどの場合、シミュレーションはオフラインの基準時間に基づいて行われ、この時点では、リアルタイム要件が制御システムに及ぼす影響については、まだ考慮されません。ここでは、プロセスモデルであるセンサ、アクチュエータおよびそれらの基礎となっている物理システムがシミュレートされます。つまりこれは純粋なソフトウェアソリューションであり、このような環境は“Software In the Loop”（SIL）と呼ばれます。

一方、このソリューションでは、革新的な制御システムの開発における最初のステップ、つまり概念のレベルでしか解決できません。イベント指向のオンラインシミュレーションでは、リアルタイムで稼働するセンサ、アクチュエータ、およ

び実際のプロセスが必要であり、オペレーティングシステムはこの時点で初めて統合されます。したがって、最初のソフトウェアプロトタイプは、Hardware In the Loop（HIL）を用いるシミュレーションで作られます。このメリットは、制御状態を一層現実的にシミュレートできることで、新しい処理をターゲット環境で、現実的な条件の下にテストし調整することができます。プロトタイピングは 2 つの段階に分けることができます。まず初期のプロトタイピングの段階では、制御アルゴリズムは物理パラメータに基づいて処理され、量子化やオーバーフローはほとんど無視されます。それらが考慮されるのは、I/O 部分（センサとアクチュエータ）のみです。次の段階では、コード実装の側面が関与してきます。最終的には、プロトタイプと製品との差はコンピュータハードウェアが変わるだけという、ほんの小さなものになります。アイデアから製品に至るまでの過程は、このように順を追って進められます。設計者はどの時点でも、この複雑なシステム全体を制御できるので、作業全体のうちの 1 つの側面だけに集中することが可能となります。

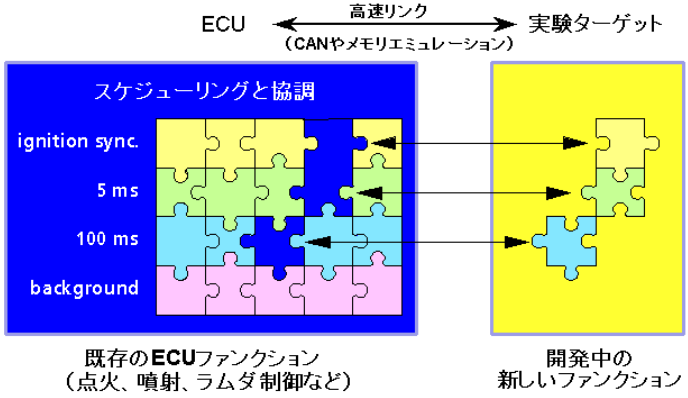


図 4-13 バイパス手法の 基本概念

プロトタイピングには、既存の ECU を利用することもできます。この場合、制御システム全体の中の一部分だけをシミュレートし、残りの部分は ECU で実行する方法（バイパス）と、ECU の信号処理システムだけを用いる方法（フルパス）とに区別されます。また、開発用 ECU でまったく新しいプログラムを実行させる方法もあります。これらの方法には 1 つの共通点があります。それは、ターゲットシステムにおいて最終製品と同等のプロトタイピングが行われる、ということです。これにより、開発の初期段階においてファンクションのレスポンスを必要な精度で調べることができ、量子化、オーバフロー、リアルタイム動作の度合いなどを、前もって把握できます。この方法を用いるための 1 つの条件は、シミュレーションとターゲットシステム（実際の ECU）のソフトウェアが同レベルで動作することであり、ASCET によりこれが実現されました。ASCET では、オンライ

ンシミュレーションと制御用コードはどちらも ETAS のリアルタイムオペレーティングシステム、ERCOS^{EK} で動作します。オンラインシミュレーションでは、データの量子化も ECU の場合と同じように行われます。

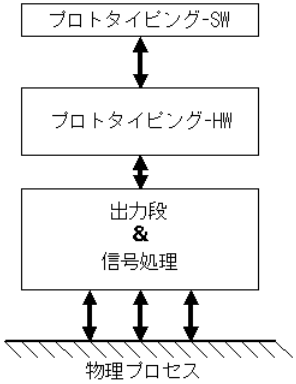


図 4-14 プロトタイピングの基本概念

環境をいかに操るかということが、プロトタイピングのもう 1 つの重要な側面であり、これには、センサやアクチュエータの基礎となっている物理システムをシミュレートすることが有効です。このようなシステムは、ECU ファンクションの離散的モデリングとは対照的な、時間連続的なシステムです。通常、このようなシステムは微分方程式により表されます。ただし、リアルタイムの処理には、非常に単純なソリューションアルゴリズムしか使用できません。というのも、この種のアプリケーションには、ステップ単位の制御はあまり適さないからです。
ASCET はこれらの方法を 1 つのツールで、つまり同じインターフェースとワークフローで、いわば 1 つのソースから提供します。たとえば、プロトタイピングを行う際に一部のコンポーネントしか使用できない場合でも、プロセスモデルを均一化されたものとして制御開発に統合できます。究極的にいえば、車両全体をオンラインシミュレーションに実装することも可能です（LabCar）。ASCET で連続性のあるシステムを構築すれば、毎日の作業に好都合です。ツール環境が変わるたびに特別なインターフェースを作成したり考え方を変えたりする必要はありません。

4.2 組込み制御システムの連続的サポート

効率の向上を左右する主な要因は、ツールチェーンの連続性です。ツールとツールの間に少しでも途切れているところがあると、開発フローの欠陥、コスト、過度の反復、あるいは手操作によるやり直しが生じることは避けられません。これは、標準的なインターフェースやフォーマットを使用することによりば回避できます。つまり、インターフェースやフォーマットは、できるだけ多くのツールメーカーがサポートしているものでなければなりません。この点について、ETAS はスタンダードの策定と普及のための活動に積極的に参加し、そのために必要なインターフェースやフォーマットをすべてのツールに取り入れています。1 社の

製品で完全なツールチェーンを提供できるのは、ETAS だけです。ETAS にとって「連続性」とは、単なるキャッチフレーズではなく、実証された「成果」であるといえます。

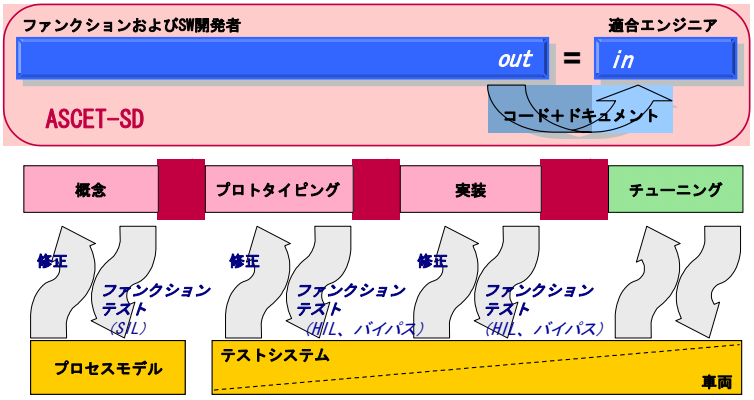


図 4-15 連続性のある開発工程

4.2.1 バイパス手法

新しい手法とアルゴリズムは、ファンクションをバイパスの形でユニークかつ実行可能な形で記述することで、効率的に導入できます。制御ファンクションの多くはある特定の製品バージョンから取り出すことができ、構築の基盤となるものがあります。このような手続きは、ASCET などの革新的なモデリング手法およびシミュレーション手法を導入する対象としては最適なものといえます。アドオン製品である ASCET-SE が、ASCET を容易に使い始められるような作業環境を提供し、また、多くの情報と共に結果が迅速に得られます。

バイパスを利用する前には、ECU 内、つまり、製品版ソフトウェアまたはそれに近いソフトウェア内の、特定のファンクションに修正を加える必要があります。この修正は、ソフトウェア制作者が行わなければなりません。この修正により、通常のファンクション呼び出しをファンクションリクエストからバイパスコンピュータに転送できます。このためにはソフトウェアスイッチが設定されます。バイパスがこのリクエストに十分に速く応答できない場合、安全のため緊急プログラムが起動され、並行して実行されている製品版ファンクションからの値が使用されます。

また、製品版ソフトウェアのデータのみを変更する場合がありますが、これに必要なのは、通常は適合のために ECU ディスクリプションファイル（ASAM-MCD-2MC）の形で提供される、アドレスと ECU の情報だけです。さらに、製品版の ECU 内で適合システムを使用してデータの測定や適合を行うこともできます。

ASCET には、一連のステート、ECU インターフェース、およびアクチュエータやセンサへのインターフェースを統合するために必要な条件が整っています。また ASCET と INCA 適合システムとを併用できることも実証済みです。

4.2.2 プロトタイピング

ASCET が適用される第 2 の領域は、プロトタイピングの分野です。ここでの目的は、新しい概念を最初から作り出すことであり、参考となるような一連のステートがない場合も少なくありませんが、プロトタイピングによって、ファンクションの単純な物理シミュレーション、量子化の影響の考察およびリアルタイム環境の推定から ECU への完全な実装までのすべてをカバーできます。どのレベルでも、ECU 環境をセンサとアクチュエータを含む形で完全にシミュレートすることが重要ですが、これは、プロセスのシミュレーションの形で、あるいは実際のハードウェアを含めた閉ループ制御によって行われます。プロトタイピングにより、制御概念の具体的な可能性が保証され、仕様に関する詳細な条件が明らかになります。

ASCET は、これらのアプリケーションに必要なコンピュータの機能を汎用プロセッサボードの形で実現し、ES1000 VME バスシステムやその他のプラグインモジュールといった外部インターフェースを提供します。

さらに、ETAS の適合システム INCA からシミュレーション環境に対してアクセスできるようにすることにより、これらのプロトタイプを小規模の車両群に装備させ、パイロットランの完成までの開発プロセス全体をカバーする経済的なソリューションを実現できます。また ETAS 独自の先進的なコード生成手法により、ターゲットのマイクロプロセッサシステムに対応する実装情報を自動コードジェネレータに与えるだけで、速やかにターゲット用コードを完成することができます。また、前もってさらに複雑なテストを行う必要がある場合は、ETAS のラピッドプロトタイピング用キャリアボード をプロトタイプ ECU として用いることもできます。

4.2.3 コードの自動生成

組込み制御システム開発において、機能モデルを実行形式のプログラムコードで表現することはかなりの難題です。物理モデリングレベルの目標と ECU レベルの目標とは非常にかけ離れています。物理モデリングでは、図式的であること、ハードウェアに依存しないこと、再利用可能なことおよび実際のデータ型（浮動小数点演算）に対応できることが求められます。また、ブロックダイアグラムやステートマシンを使用してわかり易く記述し、設計者が適合段階でもドキュメントを参照できるようにしなければなりません。一方、ECU レベル、つまり、組込みソフトウェアの実装情報には、全く別の性質が求められます。ここで必要とされるものは、既存のマイクロプロセッサと適切なメモリデバイスに合わせて最適化されたコードです。コードのサイズと実行時間が重要な要素であり、普通、データ型は整数（固定小数点演算）です。

面倒で、かつ障害を招きやすいハンドコーディングを行わずにこの 2 つの世界を結び付けるには、コードを自動生成する方法を工夫するしかありません。コードの自動生成の基本は、機能記述を物理モデリングの部分とインプリメンテーション情報の部分に分けることです。ASCET を使えば、純粋な C コードによるインプリメンテーションから新しい物理的記述方法へ、段階を追って移行できます。このシステムで生成される C コードは、他のプロジェクトでも使用できます。

このアプローチは実験環境にまで引き継がれ、量子化の影響や実行時のタイミングの問題などを早い段階で確認できます。ECU プログラムと実験環境は共通のメカニズムに基づいて構築されるので、実験システム上でアプリケーションソフトウェアを実行することができます。これによって、プログラム開発の全工程を連続的に行うことが可能となります。

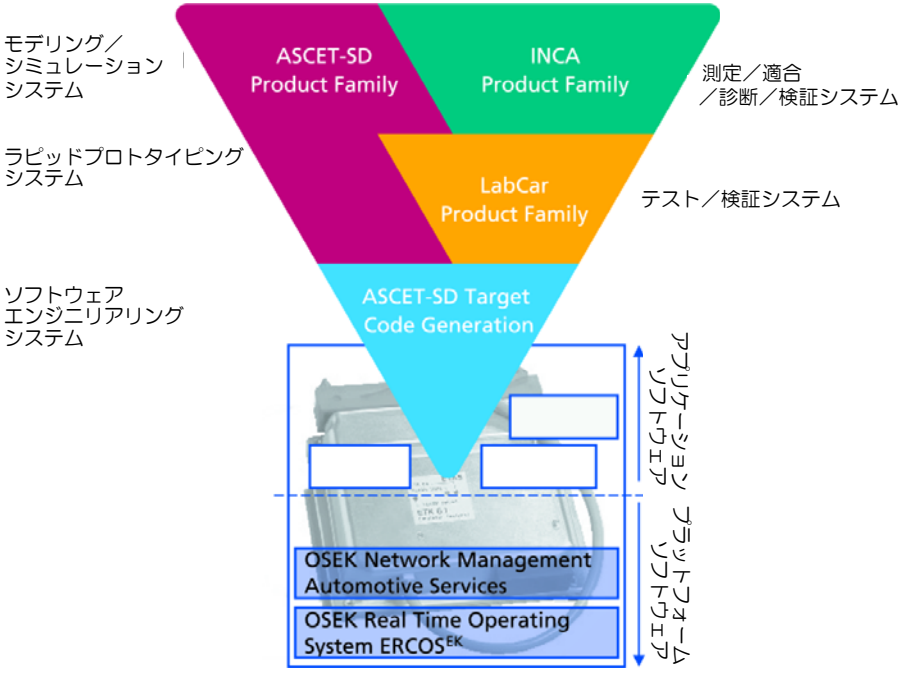


図 4-16 組込み制御システム環境でのツールの利用

4.2.4 ETAS 開発ツールの活用法

ECU 用の独自の ASCET コードジェネレータは、2 通りの利用方法があります。それは、もう 1 人のプログラマとして利用する方法と、統合ツールとして利用する方法です。前者は、たとえば大きなプロジェクトに新しい機能を追加するような場合に有用で、ASCET は優秀なプログラマとして高品質のソフトウェアを迅速に作り上げます。また ASCET を統合ツールとしてすべてのプロジェクトに適用すれば、資源の共有化やバージョン管理など、更なる品質向上と効率化を実現できます。

もう1人のプログラマとしてのASCET

一連の開発作業の中で ASCET を使用すると、従来のような仕様作成やハンドコーディングは不要になり、その結果は、C コードモジュールの形で開発製品全体に確実に反映されます。同時に、既存のコードを ASCET でステップごとに変更（リエンジニアリング）でき、その結果は実行形式のコードに反映されます。

コード生成以外の作業においては、たとえば適合作業用の ECU ディスクリプションファイル（ASAM-MCD-2MC）の生成、データスバージョンの管理などは、従来どおり手作業で行う必要がありますが、この場合も ASCET は、作業に必要な情報を生成します。

ASCET はチームでの共同作業を完全にサポートします。コンフィギュレーション管理（CM）ツールを使用するためのインターフェースが用意されており、これは、ASCET で記述された機能を共用管理する際に有用です。また CM ツールを用いない場合も、高度なインポート／エクスポート機能によりプロジェクト間のデータの共用を容易に行うことができます。

統合ツールとしてのASCET

ASCET で開発を行うと、すべてのコンポーネントを最適な形で調和させることができるというメリットがあります。各ツール間で、データ等を変換したり異なるソースを共通化したりする必要はまったくありません。ECU の機能記述、コード実装情報、およびテストケースをすべて単一の環境で作成できます。また、さまざまな情報ソースの管理の標準化も可能です。

各モジュールを統合する際に主に問題となるのは、オペレーティングシステムの設定です。各関数を呼び出す方法や情報の交換方法に関して、さまざまな定義が行われます。一定の時間で周期的に起動されるタスクと、特定のイベント（割込み）が発生したときに起動されるイベントタスクとがあり、ASCET でそれらに関するパラメータを容易に設定できます。さらにこの時点で、制御アルゴリズム（実行時のタイミングの妥当性など）を詳しく分析するための監視情報をコードに付加することもできます。

オペレーティングシステムとコンポーネント

各種タスクのプランニング（スケジューリング）は、設計エンジニアが指定する優先度に基づいて行われます。ETAS のオペレーティングシステム ERCOS^{EK} は協調的スケジューリングとプリエンプティブスケジューリングの両方をサポートします。オペレーティングシステムの主要な役割は、プロセス間で一貫性のあるデータ受け渡しを実現することであり、このためには「メッセージ」が使用されます。割込みが発生すると、現在のタスクの状態が保存され、割込み処理の終了後に復元されます。

マイクロプロセッサの周辺装置はアプリケーションにより大きく異なる可能性があるため、HW コンポーネント用のドライバは、その一部分のみがオペレーティングシステムに属しています。ハードウェアの「カプセル化アクセス」と呼ばれるものにより、部分的にはハードウェアに直接アクセスし、部分的にはオペレーティングシステム経由でアクセスします。データインターフェースという形で実現される機能の抽象化は、共通化され、周辺装置のタイプ（例、AD/DA コンバータ、PWM、CAN）により分類することができます。またオペレーティングシステ

ムと HW カプセルのレイヤの上には、適合と測定、診断とバイパスサポート用のプロトコルがあります。これらは「オートモーティブサービス」としてまとめられていて、HW カプセルやオペレーティングシステムと共に ECU ソフトウェアの基礎となります。このようなモジュール構成のモデルを ECU ソフトウェアに使用することによって、数々のメリットが得られます。たとえばエレメントのテストや交換が容易になり、またさらに、複数の開発者グループがそれぞれ別のモジュールの作業を担当することが可能となるため、稼動状態にある既存のプロジェクトを新しい ECU に速やかに移植できます。

ETAS は、ERCOS^{EK} オペレーティングシステムに加えて、適切な HW カプセルと自動車設計サービスを提供しています。このような標準パーツを用いることにより、低レベルのソフトウェア部分についてほとんど頭を悩ます必要がなくなり、制御アルゴリズムの開発に集中することができるようになりました。

4.2.5 ツールチェーンのインターフェースと標準化

標準化は工業界では非常に重要な要素です。スタンダードを無視してグローバルに事業を運営することは、今日ではとても考えられません。標準化により、機能レベルや技術レベル、および人的レベルのやりとりが確実なものになり、投資を有効に活用することができます。また、開発結果やさまざまなメーカーが製造したツールを自由に交換できるようになり、既存の情報を低コストで容易に拡張できます。

ETAS は、この「スタンダード」に主眼を置いてきました。ETAS の使命は、スタンダードを策定し、サポートすることにあります。これは、スタンダードの策定を行う組織の活動や ETAS 製品の開発に携わることで実践されています。

ETAS の高速プロトタイピングハードウェアは、業界標準の VME バスを採用しています。したがって、ETAS の実験システムはプラグインボードを追加するだけで容易に拡張できます。自動車業界では、ECU インターフェースも標準化されており、そのひとつである ASAM-MCD は、ECU と適合システムのインターフェースを定義したものです。適合、あるいは開発システムでは、ハードウェアは ASAM-MCD-1b ドライバを介してアクセスされます。ASAM-MCD-2MC ディスクリプションファイルには、ECU ソフトウェアのアドレスと変換メソッドについて必要な情報が設定されています。また、ASAM-MCD-3MC はテストスタンドからのリモート制御インターフェースを定義したものです。

開発ツールレベルのドキュメント作成とデータ交換については、自動車用スタンダードが MSR コンソーシアムから作り出されました。これにより、異なる場所で複数のパートナーがさまざまなツールを用いて開発しても、常に一貫性を保つことができます。

オペレーティングシステムのレベルでは、自動車用 ECU のスタンダードとして、OSEK が普及しつつあり、ETAS はこの規格の策定に積極的に貢献しています。オペレーティングシステムのカーネルや通信やネットワーク用インターフェースなどを対象に、HW のカプセル化や自動車用サービスなどによる標準化を進めています。

また ETAS は、ご要望に応じて他社の開発ツール（例、Matlab Integration Package）とのデータやモデルの交換などをサポートするインターフェースもご提供できます。これにより、多大なコストをかけた作業の成果を無駄にすることなく、容易に ASCET をお使いいただくことができます。

4.3 実際の ASCET の開発環境

ASCET の ECU システム開発環境は、以上のすべての必要条件に対応できるソリューションであるといえます。ASCET の革新的な機能は、戦略的な 4 つの要因である、「時間、コスト、品質、柔軟性」の各側面についての問題点の改善を支援します。ASCET パッケージにはさまざまなオプションが含まれているため、設計エンジニアはブロックダイアグラム、ステートマシン、テキスト記述および C インターフェースの中から、制御アルゴリズムに適した機能記述方法を選んで利用することができます。オペレーティングシステムに関する設定さえも、グラフィックを用いて迅速かつ簡単に行うことができます。制御プロセスをモデル化して製品に含めることができ、また連続性のあるデータベースにより、あるプロジェクトの成果物を他のプロジェクトで再利用することができます。以下の項では、ASCET の開発環境の技術的特性を中心に説明します。

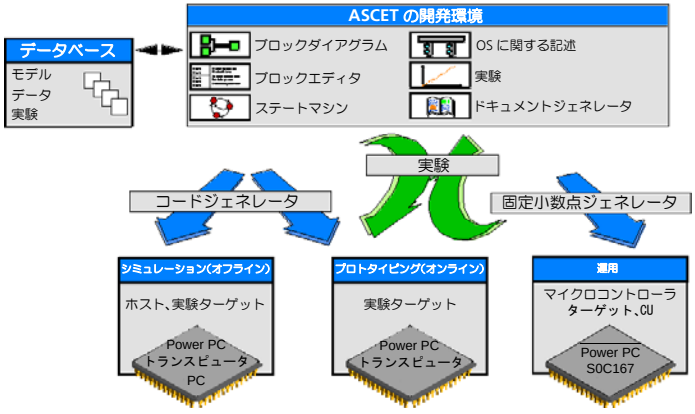


図 4-17 ASCET の開発環境の構造

機能記述レベルの下にあるのは、コード生成です。コード生成には、各種ターゲットシステムに合わせたバリエーションがあり、ユーザーのインプリメンテーション情報に基づいて行われます。物理的特性とインプリメンテーションを区分することで、ECU を直接再利用できるようになります。コード生成は、以下の各ステップに対応できます。

- 物理的特性の実験
- 量子化実験
- 実装実験
- コントローラへの実装

コード生成は、オペレーティングシステムのタスクやプロセスはもちろん、算術演算やメモリの扱いにも影響を与えます。プラットフォームに依存する事柄やプロジェクト固有の修正内容は、実用的な方法でカプセル化されます。この部分は、単なるコード生成の域を越えるものであり、ASCET は、どのようなターゲットシステムにも接続できるようにするためのあらゆる機能を備えた総合的な統合パッケージであるといえます。

ASCET のシステム開発環境においては、高度な実験システムにより、ECU プログラム内のすべてのデータに直接アクセスすることができます。プログラムをリアルタイムで実行している間でも、グラフィックを用いてデータを操作し、同時に表示することができます。このように、ユーザーはプログラムを完全にコントロールできます。

入念に設計されたハードウェアおよびソフトウェアのシステムにより、ASCET では柔軟なプロトタイピング作業を行うことができます。既存のセンサやアクチュエータを閉ループ制御に統合できるので、プロトタイプから製品へと開発を一歩ずつ進めることができます。

そしてこの革新的な技術を強力かつ経済的にサポートするのが「バイパス手法」です。システムは、ほとんどの標準的なハードウェアインターフェース（ETK および CAN）に対応しています。

もうひとつユーザーにとって好都合なのは、オープンなインターフェースです。これにより、データの再利用による投資の有効活用を実現できます。

4.3.1 制御システムの物理的機能記述

制御アルゴリズムの機能記述は、設計エンジニアの視点に基づいたものである必要があります。ブロックダイアグラムやステートマシンの作成には、実証されたグラフィックによる方法を利用します。しかし、ブロックダイアグラムを作成するよりも数学式を直接記述する方がはるかに容易である場合も多いため、ASCET では JAVA 準拠の構文によるテキスト記述をサポートしています。この項では、こういった記述方法について詳しく説明します。

まず、オペレーティングシステムの設定や制御プロセスのモデリングについて、もう少し詳しく見てみましょう。

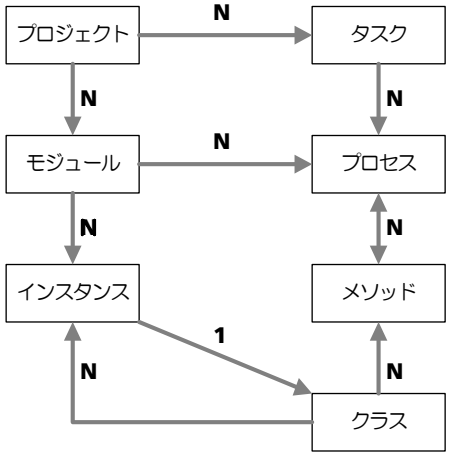


図 4-18 ASCET の記述用エレメント間の関係

ASCET での機能記述は、互いに依存する多くのエレメントで構成されます。実行形式の指定またはオペレーティングシステムの設定は、プロジェクト内で（コンポーネント単位での実験の場合は、暗黙的に生成される「デフォルトプロジェクト」内で）継続的に管理されます。プロジェクトは多くのモジュールとタスクで構成され、さらに、これらのエレメントの中にはいくつかのプロセスが含まれます。プロセスはモジュール内で定義され、その実行はタスク内でスケジュールされます。モジュールはいくつかのクラスインスタンスをオブジェクトとしてカプセル化したもので、一度しか生成されません。各インスタンスにはクラスが必ず 1 つあり、その中にはいくつかのメソッドが含まれています。モジュールとは対照的に、クラスは複数のインスタンスを持つことができます。メソッドは、引数と戻り値を内包するという点で、プロセスとは異なります。プロセス間通信はメッセージにより行われます。

クラスとモジュールは、どちらもブロックダイアグラムまたはテキストで機能記述できます。C コードのモジュールやクラスも使用できます。ステートマシンはクラスとして扱われます。

クラスは再利用可能なエレメントの代表格と言えます。ASCET は、制御技術と一般的なソフトウェア工学の分野において有効なクラスを選び、大規模なライブラリとして提供しています。

以上の ASCET のどのエレメントにも、実装情報が含まれています。この情報によって、ターゲット固有あるいはプロジェクト固有のバリエーションを扱うことができます。

ブロックダイアグラム

制御システムのブロック記述は、制御プロセスのインターフェースに基づいています。ASCET では、オブジェクト指向プログラミング言語の概念であるカプセル化されたエレメントがこのアプローチを完全なものにしています。ASCET のブロックは、情報の各アイテムをカプセル化したもので、インターフェースにより相互に接続されます。オブジェクトベースで作成されるので、容易に、しかも確実に再利用できます。ブロックダイアグラムは、制御アルゴリズムを純粋に物理的に表現するもので、これを使用すれば、エディタを使用して実装情報を容易にエレメントに追加できます。

従来のブロックダイアグラムの場合、計算の順序が正確に定義されないケースがありましたが、ASCET のブロックダイアグラムには、典型的なデータフローエレメント（例、変数、特性、算術演算子など）だけでなく、分枝などの制御フローエレメントも含まれます。制御フローは区分線と接続タイプにより、わかり易く表示されます。また ASCET では、ブロック属性を割り当てて、ブロック処理の処理順序を直接指定することもできます。こうして、シーケンシングと制御フローエレメントを使用して、複雑なアルゴリズムをグラフィック形式で正確に表現することができます。このようにすれば、複雑なアルゴリズムを扱うことができます。

また複数の企業が関与する作業においてノウハウを保護するために、アプリケーションに関するビューという概念が考案されました。ドキュメントに表示されるブロックと表示されないブロックを、ビューとして定義することができます。さらに、ASCET では、クラス、モジュールおよびプロジェクトの各レベルについて、パスワードで保護されるアクセス権も定義できます。

ステートマシン

制御システムには、階層ステートマシンが広く使われています。この設計手法は、状況に応じてさまざまな処理が必要なシステムには特に便利です。トリガ条件とステートメソッドは、ブロックダイアグラムやテキスト記述することができます。

ESDL によるテキスト記述

ESDL（**E**mbedded **S**oftware **D**escription **L**anguage）による制御アルゴリズムのテキスト記述は、C コードによるものとは異なり、複数のターゲットシステムに移植可能な物理的な機能記述です。インスタンスもクラスやモジュールにカプセル化され、そこにターゲット固有の実装情報を付加することができます。JAVA、という C によく似た言語は、普及していて容易に習得できるので、ESDL はこの JAVA に基づいています。ESDL、ステートマシンおよびブロックダイアグラムは、自由に組み合わせて使用できるため、1 つのクラスの中にブロックダイアグラムや他の形式（ESDL のテキスト指定など）の複数のインスタンスを持たせることができます。ESDL では C とは異なり、すべてのオブジェクトに対して直接そのアドレスを指定してアクセスするので、ポインタ演算は必要ありません。インスタンスが動的に生成されることはなく、すべてのオブジェクトのアドレスがコンパイル時に固定されます。これにより、ECU プログラムの整合性を早期に確認でき、使用可能メモリが十分であることがわかります。これは配列、マトリックス、特

性およびマップについても同じです。これらのオブジェクトへのアクセスもクラスのような固定的なアクセスプロトコルにより行われるので、ポインタ演算は必要ありません。

C コードの統合

ASCET による作業では、C コードは実装レベルの情報を含んだものである必要があります。つまり C コードはターゲット固有の処理が反映されたものであり、ターゲット専用のものとして管理されます。

C ソースは内部 C コードと外部 C コードの 2 通りの方法で統合できます。内部 C コードの場合、ソースは ASCET のデータベース内で ESDL のクラスと同じ方法で管理されます。一方、外部 C コードは、ユーザーの最終的なシステムに格納されるもので、他のアプリケーションにも直接使用できます。C ソースを利用できない場合には、バイナリコードを統合することもできます。

特殊なターゲットに固有なドライバがプロジェクトに含まれている場合には、C コードのブロックを使うと便利です。このブロックのインターフェースは双方向です。メソッドは C コードのブロックにより、どのクラスからでも実行できます。しかも C コードは、物理的に記述されている他のオブジェクトに対してアクセスすることもできます。ここでは実装情報の整合性がきわめて重要なので、このインターフェースは特に慎重に作成する必要があります。

オペレーティングシステムに関する設定

オペレーティングシステムへのインターフェースは、前に述べた「プロジェクト」と呼ばれるものの中に含まれます。スケジューリングされるタスクには優先度が割り当てられます。さらに各タスクには、処理順序の決定にきわめて重要なその他の属性（例、協調タスクかプリエンプティブタスクか、周期的に実行されるか外部イベントで起動されるか、それともプログラムの最初に 1 回だけ起動されるかなど）があります。オペレーティングシステムに関する設定は、これらを基礎として、その他の情報も考慮しながら行います。タスク内で実行されるプロセスは、メッセージを介して情報を交換しあいます。各モジュールのメッセージは、

同じ名前のもの同士がつなぎ合わされます。コードジェネレータが、この情報の整合性（複数ポイントでのメッセージの使用、グローバル変数の扱いなど）を調べます。

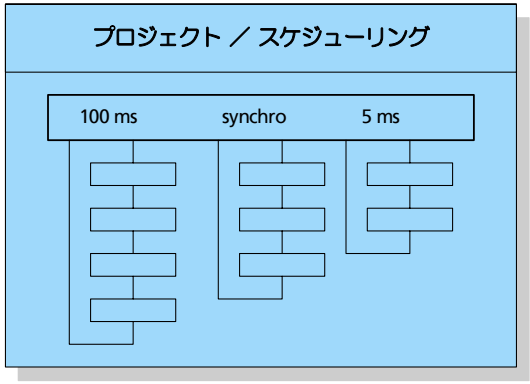


図 4-19 プロジェクト内のプロセスとタスク

オペレーティングシステムに関する設定はシンプルなエディタで視覚的に行います。この方法を利用すれば、システム全体について連続性のある概要を捉えることができます。スケジューリングの変更も非常に容易かつ迅速に行うことができます。また、ETAS のオペレーティングシステム、ERCOS^{EK} はコンパイル済みのライブラリが提供されているため、リンク処理以外では、オペレーティングシステムに関する設定変更には、ほとんど手間がかかりません。

制御プロセスのモデリング

今日では、プロセスのモデリングを行わずに複雑な制御アルゴリズムを作成することはあまり考えられません。その点において ASCET には、制御プロセスとそのプロセスモデルを同じシステムで開発できる、という大きなメリットがあります。これにより、データ変換やシミュレータの連結に余計な時間を費やすことはなくなります。小規模プロジェクトでは、プロセスと制御が単一のソースから作りだされるため、1 人のエンジニアが両方を担当することができます。

ASCET は、他のシステムを統合できるオープン性も備えています。たとえば、ETAS は Matlab へのリンクを実現しました。これにより、Simulink モデルを ASCET の機能記述と一緒にシミュレートできます。

以上のような手法は開発の効率と効果を高める大きな一歩となります。これは特に、実験室においてモデルと実験用ハードウェアによって車両をシミュレートする、という場合などに有効で、莫大な時間を節約できます。安全限界に関する分析も、少ない費用で作業机の上で安全かつ速やかに行うことができます。運転者が車両に乗り込む必要があるのは試乗の時だけです。ETAS は、この手法を多くのシステム用に提供しています。

4.3.2 インプリメンテーション（実装情報）とコード生成

最終製品の ECU 用のコードの自動生成は、開発を効率的に行う鍵ですが、ASCET はこれについてスタンダードを設定しました。固定小数点演算だけでなく、オペレーティングシステムインターフェースを浮動小数点プロセッサ用に設定することも必要です。メモリ管理を最適化し、カプセル化されたハードウェアを統合する必要があります。また、浮動小数点システムでは、単精度と倍精度のデータを区別する必要があり、少なくともシミュレーションにおいてはこれが非常に重要です。

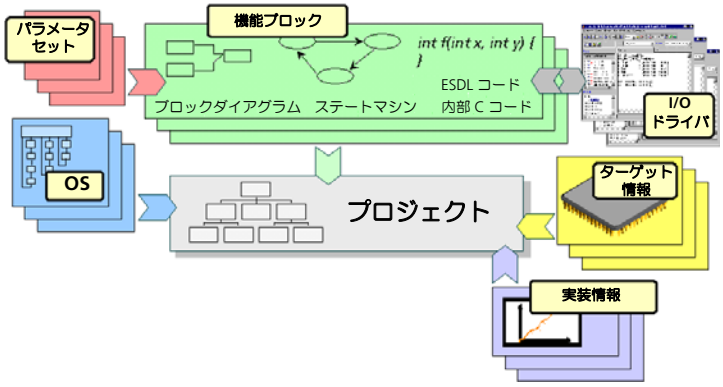


図 4-20　ASCET におけるコード自動生成の基本概念：プロジェクト作成

ASCET のインプリメンテーション（実装情報）は、データ型、値の範囲およびメモリ格納情報で構成されています。これらのインプリメンテーション情報と物理データ記述の関連性は、変換式によって定義されます。メモリ格納情報以外に、関数をインラインで実行するかどうか、あるいは特性分析のためにユーティリティを使用するかどうかについての定義も行われます。

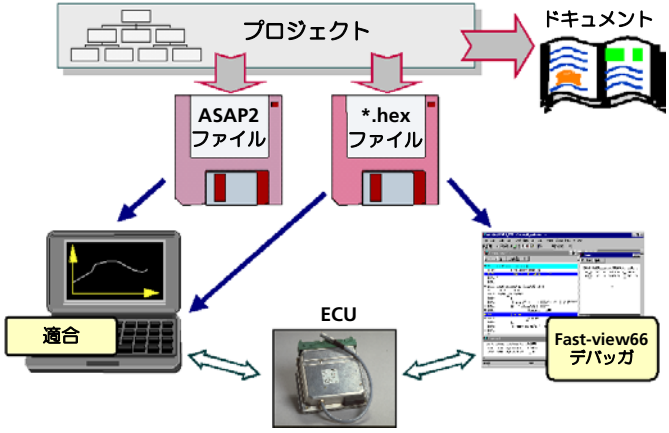


図 4-21 ASCET における自動コード生成の基本概念：結果出力

コード生成の処理は一本道ではありません。生成される C コードと並行して、ASAM-MCD-2MC ディスクリプションが必要です。これは、適合／測定システムに必要なアドレス情報を提供するもので、この情報は、コード生成後に MAP ファイルを用いて生成されます。これを行う機能は ASCET-SE に含まれていて、また、先に ASAM-MCD-2MC ファイルを作成して ASCET に読み込ませ、それを元にしてデータモデルを作成することもできます（既存プログラムのリエンジニアリング）。

アルゴリズム

ブロックダイアグラム、テキスト記述、およびステートマシンをターゲット固有の C コードに自動変換する処理は、共通の中間レイヤで実行され、そこではインプリメンテーション情報を使用して最適化されたターゲットシステムへのアルゴリズムのマッピングが行われます。ここでは、複雑なロジックの展開の他に、算術演算に特有の問題が生じます。

```
a = b
```

整数コード生成においても、上のような 2 つの変数による単純な代入式をコード変換する場合も、インプリメンテーションが異なると一筋縄ではいきません。

また、a と b を符号なしの 8 ビット変数（値の範囲は 0 ～ 255）として、次の式を考えてみましょう。

```
a = 2 * a_impl, b = 3 * b_impl
```

これに、以下のような単純な代入が続きます。

```
a_impl = 3 * b_impl / 2
```

ここでは、一連の演算で最大限の精度を得られるように注意しなければなりません。除算を先に行ってしまうと、整数計算が原因で変換式の効果が失われてしまい、結果は約 50%の誤差が発生してしまいます。

```
a_impl = (3 / 2 )* b_impl = b_impl
```

桁あふれの問題も考慮する必要があります。つまり、先に 3 を掛けると、`b_impl` が 255 / 3 = 85 より大きくなるとけたあふれが起こります。同様に、アンダーフローや丸め誤差についても注意が必要です。初めに 2 で割ると、これは右シフト演算と同じことなので、最後のビットが失われます。そのため、`b_impl` の値が 1 の場合と 0 の場合の違いがなくなります。どちらの場合も、結果は `a_impl` の値が 0 になり、ひいては `a` の値も 0 になります。

実際、`a = b` という代入が意味をなすのは、a と b の物理範囲が同一（この場合は最大 0 ～ 510）の場合だけです。ですから、`b_impl` の最大値は、510 / 3 = 170 と推定されますが、それでもけたあふれが発生する可能性があります。それを防ぐために、コード生成で場合分けを行うことも可能です。つまり、`b_impl` の値が 170 以下なら先に乗算を行い、170 より大きければ先に除算を行うようにするのですが、これだと、コードが余計に必要になります。そのため、ここでは値の範囲内で、最大 1.5 の精度で誤差を受け入れる必要もあります。

複雑な制御や式はもちろん、少数のオペランドを伴う通常の算術演算でも、状況はこのように非常に複雑になり得ることは明らかです。ASCET のコード自動生成機能は、この種の問題からユーザーを解放してくれます。

メモリの扱い

ECU のアーキテクチャにより、各種目的に使用するメモリ領域が異なります。プログラム用に確保される領域もあれば、適合データを格納する領域もあります。ROM、RAM、およびフラッシュメモリを物理的に区別する必要もあります。多くの ECU では、ビット格納専用の領域が確保されたり、または偶数アドレスだけに値を格納できる場合もあります。また、外部との間の入出力として確保されるメモリ領域もあります。ASCET は、この情報を継続的に管理します。ECU のアーキテクチャに基づいてインプリメンテーションを記述することができるので、どのような物理的条件にも適応させることができます。

すべての標準データ（たとえば、測定変数や適合変数）には、デフォルトのメモリ領域が自動的に割り当てられます。コード自動生成ファンクションはこれらのインプリメンテーションを C コードの pragma 文で表現します。ここでもやはりユーザーは、煩雑でしかもエラーを招きがちなプログラム管理業務から解放されます。

オペレーティングシステムに関する設定

オペレーティングシステムに関する設定も、ASCET で行えば非常にシンプルなグラフィカルユーザーインターフェースによってプロジェクトに必要な情報を指定することができます。この場合も、C コードが自動生成されます。

コード自動生成の主な役割は、メッセージという手段による各種プロセス間の通信を最適化することです。割込みが発生した場合には、プロセスが中断された後、再び整合性のあるデータで処理を続行できるように、すべてのメッセージを保存する必要があります。これを実現するには膨大な労力がかかりますが、実際には、すべてのメッセージを保存する必要がない場合が多く、中断されている間に整合性が損なわれる可能性のあるデータだけを保存すれば十分です。コード自動生成ファンクションは、こういった高度の最適化を実現します。

この際、複雑な割込みオプションはコード生成ファンクションで自動的に分析され考慮されるので、ユーザーは多くの作業から解放されます。これで、バックアップ操作を忘れてもエラーは起こらなくなります。開発段階でプロセスやタスクの構成が変更になる可能性がある場合でも、複雑なプログラム管理に対するユーザーへの負担は大幅に軽減されます。このように、ASCET は利便性と信頼性の両方を提供します。

プラットフォームへの依存性とプロジェクト固有の条件

ASCET のコード自動生成の性能が優れていることは、多くのユーザーの方の製品開発における成功例にだけでなく、将来のための拡張性という面にも表れています。この鍵は、オープンで適応性のあるインターフェースで、たとえば、他社のオペレーティングシステムを統合できることや、プロジェクト固有の条件（例、命名規則や顧客固有の開発工程など）をうまく反映できることなど、多くの例で実証されています。

開発作業を物理モデリングとインプリメンテーションの記述に分けたことが基礎となって、広範にわたる応用が可能になりました。プラットフォーム固有の特性をインプリメンテーションの中心部分にカプセル化できるので、それを一度に変更することができます。これに、コード自動生成機能用のさまざまな設定オプションが追加されます。コード生成規則は ASCET の中央の中間レイヤ上に構築され、ASCET-SE 内で使用されて、特定の条件に適応させることが可能となります。これにより、かつてないほど優れた方法でコード自動生成ファンクションを変更できるようになりました。

4.3.3 ASCET によるプロトタイピング

ASCET では、プロトタイプレベルでの結果を迅速に作り出し、さらにそれを迅速に製品化することができます。このプロトタイピング手法を成功させたのは、高機能なハードウェア製品群と、それをサポートする実験環境です。この項では、これについて詳しく説明します。

あらゆる要求に対応する実験環境

ASCET の実験環境においては、開発オブジェクトに変更を加えなくても、それらのオブジェクトについてさまざまな分析を行うことができます。この実験環境は、実行される実験に与える値を迅速かつ明解に定義して生成し、データの記録や分析を行い、変数を変更することのできる、汎用的な作業環境です。シミュレーションは任意の時点で開始／終了できます。各種設定、最適化したデータセット、ウィンドウ構成などを個別に保存できるので、同じ環境を何回でも使用することができ、また各モデルについて目的別に複数の実験を管理することもできます。

さまざまなグラフィック表示やテキスト表示機能、測定と適合のためのサブウィンドウがあり、それらを自由に組み合わせて画面に配置できます。ASCET の実験環境では、このような優れた機能を、オフライン実験、およびリアルタイムで実行されるオンライン実験の両方に利用できます。

ASCET では、最初に制御アルゴリズムを物理実験によって分析した後に、量子化とインプリメンテーションの検証も同じ作業環境で行うことができます。このように、システムを順を追って調整し、最終的なソフトウェア製品に仕上げることができます。エラー監視用の変数を実験に組み込んで、デバッグに活用することもできます。

この実験環境では、新しく作成されたモジュールまたはクラスを、1 つずつ順にシミュレーション環境に組み込んでいくことができます。またオペレーティングシステムの動的な設定を実験環境内で直接行うことができるので、リアルタイムに実行された現実的な結果を迅速に得ることができます。

ASCET の実験で使われるエレメントの表示と適合の機能は、ETAS の INCA 測定・適合システムで用いられているものと同じです。つまり、開発初期の概念の段階から最終製品の段階に至るまで、一貫した開発環境を用いることが可能となるわけです。

シミュレーションシステム：ハードウェアとソフトウェア

プロトタイピングの手法を支える大きな要素の 1 つは、実験用ハードウェアです。その中核となるのがシミュレーション用プラットフォーム ES1000 システムで、これは、拡張可能なモジュラー構成となっている VME バスシステムです。このシステムに組み込まれた高性能なプロセッサボード（PowerPC 搭載）が新しい制御アルゴリズムを実行し、さらにさまざまな拡張ボードを追加することができます。

シミュレーションプラットフォームのオペレーティングシステムは、最終製品である ECU 内のものと同じ ERCOS^{EK} です。ですから、プロトタイピングは非常に現実に近い環境で行われ、必要な分析をすべてリアルタイムで実行できます。

図 4-22　ES1000 汎用 VME バスシステム

シミュレーションを行う際、さまざまなオプションボードを ES1000 システムのプラグインボードとして利用することができます。プラグインボードとして利用できるボードには、外部の一般的なセンサやアクチュエータを利用するための汎用ボードから、プロジェクト固有の特別な信号処理を行うボードまで、幅広い種類のものがあります。

既存のセンサやアクチュエータを閉ループ制御に統合する

ETAS は、センサやアクチュエータ用によく使われるすべてのインターフェースについて、ハードウェアとソフトウェア両面でのソリューションを提供します。VME バスボードとしては、以下プラグイン I/O ボードが用意されています。

- AD/DA 変換
- PWM 信号の入出力
- CAN インターフェース
- LIN インターフェース
- デジタル I/O
- FPGA

さらに、温度の測定と排気データの記録などに用いることができる、特別なソリューションも用意されています。

PC インターフェースにはイーサネットが使用され、モバイルのラップトップコンピュータで作業することができます。ASCET は、効率的なプロトコルを用いて実験ハードウェアとのデータ交換を行います。これによりハードウェアレベルでの閉ループの実行が妨げられることはありません。

4.3.4 バイパス

バイパス手法を利用するための物理インターフェースとして、ETK によるメモリエミュレーションと CAN インターフェースの 2 つが用意されています。これ以外のハードウェアインターフェースも、ご要望に応じてカスタマイズできます。すべてのバイパスシステムは ECU 内の一部のファンクションについて変更や開発を行うためのものです。バイパスを適用するためには、そのファンクションの修正が必要で、これには、ファンクションのスケジューリングの変更が含まれます。その際、そのファンクションの処理時間をあらかじめ把握し、通信エラーなどの発生に備えて安全対策を講じておく必要があります。たとえば、通信エラーによって処理時間のタイムアウトが発生した場合は、緊急プログラムを起動させるか、あるいはデフォルトのデータを代用させることも考えられます。

バイパス手法は主として、ノウハウの保護の理由で公開できないファンクションが含まれる共同開発プロジェクト向けの対応策として考案されました。設計エンジニアに対しては一部のインターフェースしか解放されないため、エンジニアは他の煩雑な部分を考慮する必要がありません。しかも、システム全体の主要部分は常に安定しているので、新しいファンクションの開発に完全に集中でき、その結果、非常に有効で経済的なソリューションを作り出すことができます。

ETK インターフェースによるメモリエミュレーション

エミュレータテストプローブ（ドイツ語の頭字語：ETK）は、ETAS が開発した独自の ECU 用インターフェースです。これは、ECU のメモリを DPRAM によって全体的または部分的にエミュレートする機能を持っています。ECU は一方の側から ETK に直接アクセスし、ユーザーは PC 側から ETK を意識せずに ECU の値にアクセスすることができます。このメモリエミュレーションを行うには ETK とプロセッサの距離を非常に短くする必要がありますが、今日の ETK は非常に小型化されているので、ECU 内に組み込むことも可能です。

ES1000 システムのプラグインボードとして、ETK との接続に使用される専用のインターフェースボードが用意されています。また ETK は多くの異なるタイプのプロセッサに対応してさまざまな機種が用意されているので、適合のための汎用的なソリューションであるといえます。

CAN インターフェース

コスト的な理由や ECU の技術的制約などの理由によって ETK を使用できない場合、代わりのソリューションとして CAN インターフェースを利用できます。CAN インターフェースを用いる場合、ECU ハードウェアに変更を加える必要がなく、ほとんどすべての ECU に適用できます。性能面では ETK には及ばず、しかも多くの CAN メッセージが必要となる、というマイナス面もありますが、経済的で手軽であることから、ASCET と共に広く利用されています。

4.3.5 再利用可能でオープンなインターフェース

再利用については、モジュール化、オブジェクト指向、インターフェースの互換性などの面から何年にもわたって議論されてきましたが、ASCET では、再利用可能でオープンなインターフェースが実現されました。これは、実際のプロジェクトにおける開発手順や開発結果を入念に分析し、これらの考察結果をツールの特性として反映したことによる成果です。

その成果の 1 つとして、ETAS は制御アルゴリズムを記述する際に使用されるブロックライブラリを構築しました。これらのブロックは実用的で優れていることがすでに実証されています。

再利用可能な実験はこのライブラリにリンクされているので、新旧の開発作業において再現性のある結果が得られます。適合ツールやテストツールとのインターフェースにより、すべての作業手順をシステム全体の開発工程に組み込むことができます。また ETAS の他のツールに対するインターフェースだけでなく、一般的に使用されている ASAM や MSR などの標準インターフェースもサポートしています。

再利用を支えるデータベース

ETAS の再利用についての概念の基本となるものは、ASCET のデータベースシステムです。このシステムは最初から ASCET カーネル内に構築されていて、ASCET の土台となっています。このデータベースにおいてはすべてのデータを個別の作業環境に確実に保存できるうえ、無許可な操作、あるいは誤操作やデータ破損な

ども防ぐことができます。最終的にこのデータは、特別な保護を必要とする貴重な成果物となるため、ASCET のデータベースは、ユーザーやチームによる日々の作業を効率的に進めるための適切なメカニズムによって管理されます。

コンフィギュレーション管理ツールを使用して行うプログラム管理とデータ管理

多彩な開発作業の流れを管理するため、あるいは大規模なチームが複雑なソリューションについて共同で作業できるようにするために、さまざまなコンフィギュレーション管理ツールが使用される場合があります。そのため ASCET には、各種コンフィギュレーション管理ツールと連携して機能できるようにするためのインターフェースが用意されています。この際、通常 ASCET はコンフィギュレーション管理ツールのクライアントとして稼働しますが、ASCET をサーバーとして使えば、ASCET のデータベース機能を存分に利用してコンフィギュレーション管理ツールの管理メカニズムの細部を修正することもできます。JAVA のインターフェースを修正するだけで ASCET をサーバーとして使用でき、これによって革新的で複雑な管理業務を実現できます。

コンフィギュレーション管理ツールをご利用になる場合は、ETAS までお問い合わせください。用途に合わせたソリューションをご提供いたします。

4.4 ASCET ソフトウェアの構成

ASCET ソフトウェアファミリは、4 つの製品で構成されています。各製品が、それぞれ異なる開発工程での作業をサポートします。

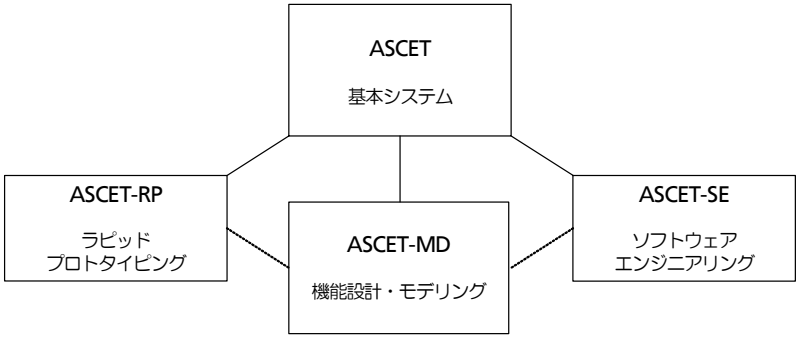


図 4-23 ASCET のモジュール構成

以下に、個々の製品の機能について簡単にまとめます。

ASCET 基本システム

ASCET 基本システムは ASCET ファミリの中核となる部分です。他の製品をインストールするには、まずこの基本システムをインストールする必要があります。

ASCET-MD （モデリング・デザイン）

ASCET-MD（**ASCET M**odeling & **D**esign）を使用すれば、ブロックダイアグラム、ステートマシン、ESDL、C プログラミング言語でモデルを記述することができます。以前のバージョンの ASCET と同様、モデルの記述と管理、さらにオフライン環境でのシミュレーション実験も行えます。

同じバージョンの ASCET-MD は、1 台の PC に 1 システムのみインストールできます。

ASCET-RP （ラピッドプロトタイピング）

ASCET-RP（**ASCET R**apid **P**rototyping）にはラピッドプロトタイピングに必要なすべての機能が含まれています。

ASCET-RP では、モデリングされたコンポーネントの内容をモニタすることができますが、モデルやモデルエレメントの作成と管理には ASCET-MD が必要です。

同じバージョンの ASCET-RP は、1 台の PC に 1 システムのみインストールできます。また ASCET-MD、ASCET-RP、および 1 つまたは複数のマイクロコントローラ用 ASCET-SE をすべて同じ PC にインストールすることも可能です。

ASCET-SE （ソフトウェアエンジニアリング）

ASCET-SE（**ASCET S**oftware **E**ngineering）は ECU コードの生成に必要なすべての機能が含まれています。

ASCET-SE では、モデリングされたコンポーネントの内容をモニタすることができますが、モデルやモデルエレメントの作成と管理には ASCET-MD が必要です。

ASCET-SE は、いくつかのマイクロコントローラターゲット用のものが用意されています。他の ASCET 製品とは異なり、同じ PC 上に異なるターゲット用の複数の ASCET-SE をインストールでき、それらは ASCET-MD と ASCET-RP と同じ PC にインストールすることができます。

5 ASCET の基本操作

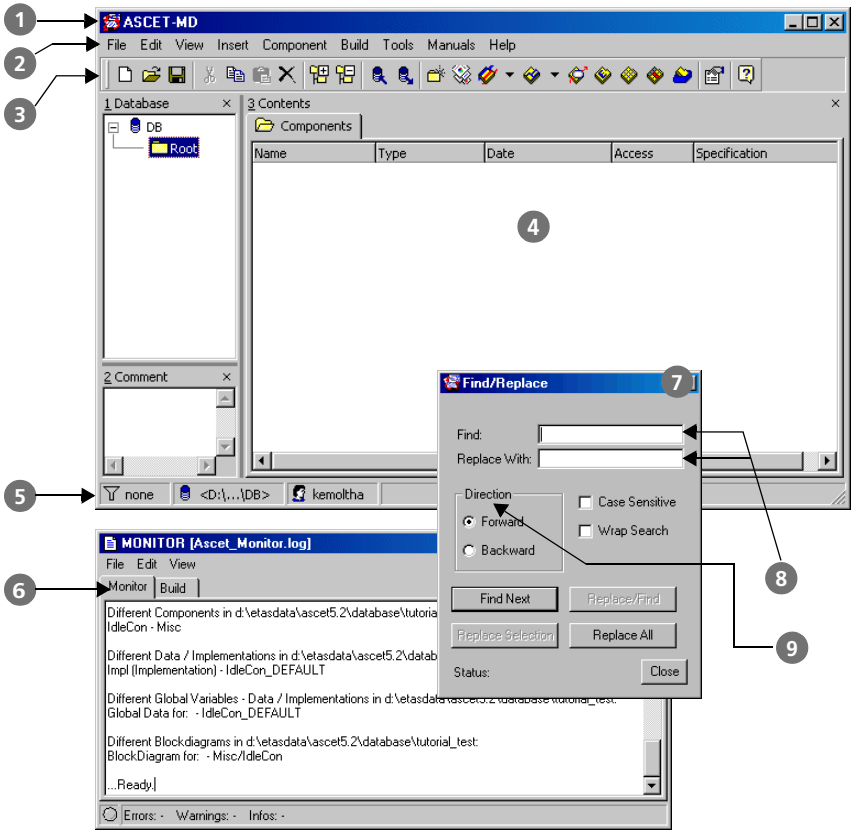
本章では、ASCET のユーザーインターフェースの一般的な情報、つまりウィンドウとメニューの構造、マウスやキーボードによる操作方法、およびヘルプ機能について説明します。

ほとんどの操作は Windows の標準オペレーションに基いていますが、ウィンドウ内の各制御エレメントの機能など、ここでしか説明されていない情報も含まれますので、本章はよくお読みいただくことをお勧めします。

ASCET の開発においては、キーボードでの迅速な操作を可能にすることに重点が置かれてきました。キーボード操作に関しての特殊な機能や Windows との違いについては、213 ページの「キーボード操作」を参照してください。

5.1 ウィンドウの構造

ASCET の各ウィンドウは、以下のような構造になっています。



- タイトルバー（1）
- メニューバー（2）
- ツールバー（3）
- ウィンドウエリア（4）
- ボトムバー（5）
- タブ（6）
- ダイアログボックス（7）
- フィールド（8）
- フィールドタイトル（9）

5.2 ツールバー

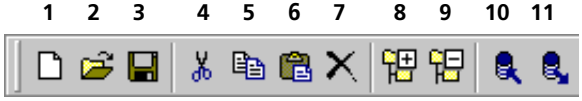
ツールバーには、よく使われるコマンドのボタンやコンボボックスが用意されています。ボタンをクリックするだけで、簡単にコマンドを実行できます。

注記

ボタンに割り当てられたコマンドは、すべてメニューからも実行できます。

ツールバー上のボタンはマウスの動きに反応します。ポインタをボタンの位置に合わせて 1 秒間そのままにすると、そのボタンの脇にテキストボックスが開き、ボタンの機能と対応するキーボードコマンドが表示されます。

5.2.1 コンポーネントマネージャのボタン



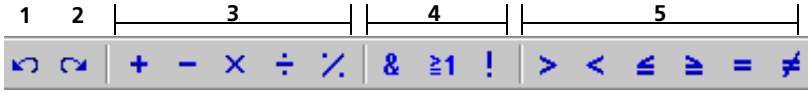
1. New - 新規作成（新しいデータベースの作成）
2. Open - 開く
3. Save - 保存
4. Cut - 切り取り
5. Copy - コピー
6. Paste - 貼り付け
7. Delete - 削除
8. Expand all - すべて展開

9. Collapse all - すべて省略
10. Import - インポート
11. Export - エクスポート

12. Insert Folder - フォルダの挿入
13. Insert Project - プロジェクトの挿入
14. Insert Module - <Type> - モジュールの挿入 < タイプ >
15. Insert Class - <Type> - クラスの挿入 < タイプ >
    矢印ボタン（▼）14a および 15a でオブジェクトタイプを選択します。
16. Insert State Machine - ステートマシンの挿入
17. Insert Enumeration - 列挙型の挿入
18. Insert Boolean Table - 論理テーブルの挿入
19. Insert Conditional Table - 条件テーブルの挿入
20. Insert Container - コンテナの挿入
21. Options - オプション
22. ? - “About ASCET”ウィンドウ を開く
    インストールされている ASCET 製品ファミリについての情報が表示されます。

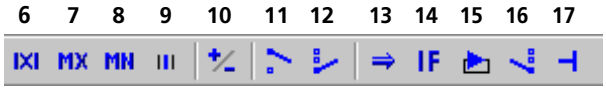
23. Search < 検索対象 > - データベース内の検索対象を選択して、指定の文字列を検索
24. 検索文字列の入力フィールド

5.2.2 ブロックダイアグラムエディタのツールバー



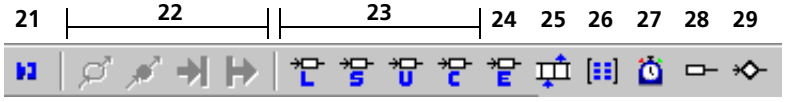
1. Undo - 元に戻す
2. Redo - やり直し
3. Addition, Subtraction, Multiplication, Division, Modulo - 算術演算

4. And, Or, Not - 論理演算
5. Greater, Less, Less or Equal, Greater or Equal, Equal, Not Equal - 比較演算



6. Abs - （入力値の絶対値を返します）
7. Max - （最も大きな入力値を返します）
8. Min - （最も小さな入力値を返します）
9. Between - （入力値が指定範囲内であるかどうかを判定します）
10. Negation - （入力値の符号を反転します）
11. MUX - マルチプレクサ
12. Case
13. If-Then
14. If-Then-Else
15. While
16. Switch
17. Break - （プロセス／メソッドの実行を中断して呼び出し元に戻ります）

18. 演算子の入力数を選択するコンボボックス
19. ビューを選択するコンボボックス
20. ズーム倍率を選択するコンボボックス

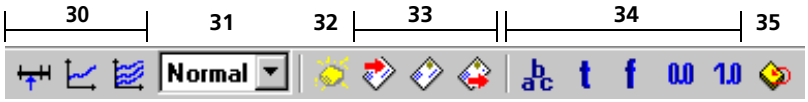


21. Connect - 接続
22. State, Junction, Input, Output - ステートマシン用エレメント（ステートマシン編集時にのみ有効）
23. Logic, Signed Discrete, Unsigned Discrete, Continuous - 変数
24. Enumeration - 列挙型データ
25. Array - 配列
26. Matrix - マトリックス

27. dT - システムパラメータ

> 注記
>
> dT という名前はシステムパラメータ用に予約されているため、ユーザーがこの名前のエレメントを作成することができません。大文字と小文字は区別されないため、DT、または dt という名前も使用できません。

28. Continuous Parameter - 連続型パラメータ
29. Implementation Cast - インプリメンテーションキャスト

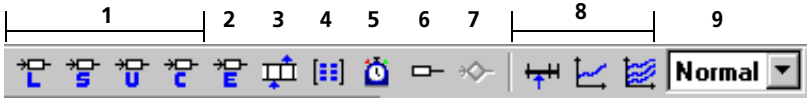


30. Distribution, One D Table Parameter, Two D Table Parameter - 特性カーブ／マップ
31. 特性カーブ／マップのタイプを選択するコンボボックス
32. Ressource - リソース
33. Receive, Send Receive, Send - メッセージ（モジュール編集時にのみ有効）
34. String, True, False, 0.0, 1.0 - リテラル
35. Self - 編集中のオブジェクト自身への参照

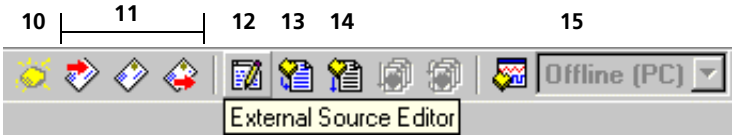
36. Hierarchy - 構造ブロック
37. Comment - コメント
38. Generate Code - コード生成
39. Open Experiment for selected Experiment Target - 選択されている実験ターゲット用の実験環境を開く

番号の付いていない操作エレメントは、ブロックダイアグラムエディタでは常に無効になっています。

5.2.3 C コードエディタ／ ESDL エディタのツールバー

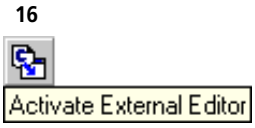


（1）～（9）は、C コードエディタと ESDL エディタの両方で使用できます。ただしボタン（7）だけは C コードエディタでは使用できません。これらのボタンはブロックダイアグラムエディタのボタン（23）～（31）に相当します（83 ページ参照）。



ボタン（10）、（11）、（13）、（15）も、両方のエディタに共通して使用できます。これらのボタンはブロックダイアグラムエディタのボタン（32）、（33）、（38）、（39）に相当します（83 ページ参照）。

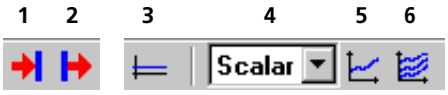
12. External Source Editor - 外部ソースエディタを開く（C コードエディタでのみ有効）
14. Compile Generated Code - 生成されたコードをコンパイルする（C コードエディタでのみ有効）



16. Activate External Editor - 外部のテキストエディタを開く
    このボタンは、両エディタのボトムバー内にあります。

5.2.4 CT ブロックエディタのツールバー

CT ブロックは C コード、ESDL、またはブロックダイアグラムで記述できます。これらのエディタで CT ブロックを編集する際には以下の CT ブロック用ボタン（およびコンボボックス）が使用できます。



1. Input - 入力
2. Output - 出力
3. Constant - 定数
4. パラメータの型を選択するコンボボックス
5. One D Table Parameter - 特性カーブ
6. Two D Table Parameter - 特性マップ

ボタン（7）と（9）は、ブロックダイアグラムエディタのボタン（38）と（39）に対応します（85 ページ参照）。

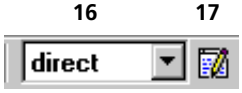
8. Compile Generated Code - 生成されたコードをコンパイルする（CT ブロック用 C コードエディタでのみ有効）

番号の付いていないボタンは、CT ブロックエディタでは常に無効になっています。

以下のボタンは、ESDL エディタまたは C コードエディタを使用する場合にのみ有効です。

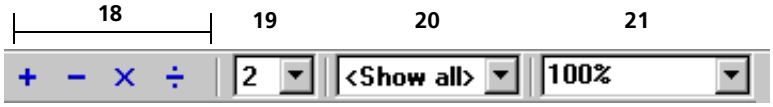
10. Discrete State - 離散ステート
11. Continuous State - 連続ステート
12. Steplocal - ステップローカル変数
13. Parameter - パラメータ（型はコンボボックス（4）で選択します）
14. Dependent Parameter - 依存パラメータ（型はコンボボックス（4）で選択します）
15. Activate External Editor - 外部のテキストエディタを開く
    このボタンは、エディタのボトムバー内にあります。

以下の操作エレメントは C コードエディタで CT ブロックを編集する際にのみ有効です。

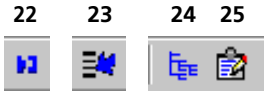


16. ブロックの挙動（直接／間接）を選択するコンボボックス
17. External Source Editor - 外部ソースエディタを開く（C コードエディタでのみ有効）

以下の操作エレメントはブロックエディタで CT ブロックを編集する際にのみ有効です。

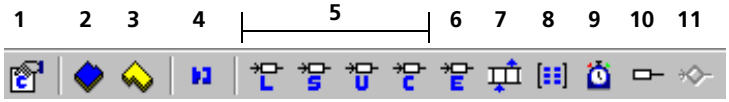


18. Addition, Subtraction, Multiplication, Division - 算術演算
19. 演算子の入力数を選択するコンボボックス
20. ビューを選択するコンボボックス
21. ズーム倍率を選択するコンボボックス



22. Connect - 接続
23. Global Parameter - グローバルパラメータ（コンボボックス (4) で型を選択）
24. Hierarchy - 構造ブロック
25. Comment - コメント

5.2.5 プロジェクトエディタのツールバー



1. Specify Code Generation Settings - コード生成に関する設定（“Settings” ダイアログボックスの “Code” タブが開きます）
2. Edit Data - データの編集（プロジェクトのデータエディタが開きます）
3. Edit Implementation - インプリメンテーションの編集（プロジェクトのインプリメンテーションエディタが開きます）

ボタン（4）～（11）はブロックダイアログエディタのボタン（21）と（23）～（29）に相当します（83 ページ参照）。

ボタン（12）、（13）、（15）、（16）はブロックダイアログエディタのボタン（30）、（31）、（36）、（37）に相当します（83 ページ参照）。

14. Send Receive Message - メッセージ送受信

番号の付いていないボタンは、プロジェクトエディタでは常に無効になっています。

ボタン（17）はブロックダイアグラムエディタのボタン（38）に相当します（85 ページ参照）。

18. Compile Generated Code - 生成されたコードのコンパイル
19. Build Executable Code - 実行コードのビルド
20. Rebuild Executable Code - 実行コードの再ビルド

21. Transfer Project to selected Experiment Target - 選択された実験ターゲットへのプロジェクトの転送（プロジェクトをコンボボックス（25）で選択された実験用に変換して出力します）

    このボタンは、ターゲットオプションで `ES1130` または `ES1135` がターゲットとして選択されていて、さらにコンボボックス（25）で `INCA` または `INTECRIO` が選択されている場合にのみ有効です。

22. Open Experiment for selected Experiment Target - 選択された実験ターゲット用の実験を開く（コードを生成し、コンボボックス（25）で選択されたターゲット用の実験を開きます）

    このボタンは、コンボボックス（25）で `INCA` または `INTECRIO` が選択されている場合は無効です。

23. Reconnect to Experiment of selected Experiment Target - 選択された実験ターゲットの実験への再接続（コンボボックス（25）で選択されたターゲット上で実行されている実験に再接続します）

    このボタンは、コンボボックス（25）で `Offline (RP)` が選択されている場合は無効です。

24. Experiment Target - 実験ターゲット（現在ターゲットオプションで選択されているターゲットについて選択可能な実験環境が表示されます）

25. "Formulas" タブの更新
26. Add missing Formulas - 不足している変換式の追加

    現在アクティブなインプリメンテーション内で使用されている変換式がチェックされ、未定義のものが見つかると、その名前を持つ一次変換式が追加されます。

5.2.6 オフライン実験用のツールバー

1. Exit to Component - 実験ウィンドウを閉じてコンポーネントエディタを開く（ボタンの絵は、実験を行っていたコンポーネントのタイプに応じて異なります）
2. Load Environment - 実験のロード（あらかじめ変数が割り当てられた測定・適合ウィンドウをロードします）
3. Save Environment - 実験の保存
4. Save Environment As - 実験に名前を付けて保存
5. Stop Offline Experiment - オフライン実験の終了

6. Start Offline Experiment - オフライン実験の開始
7. Pause Offline Experiment - 実験の一時停止（ボタン（6）または（8）で続行します）
8. Step Offline Experiment - オフライン実験のステップ実行
9. ステップサイズの入力フィールド

10. Open CT Solver - CT ソルバを開く
    このボタンは、CT ブロックまたはハイブリッドプロジェクトの実験を行っている場合のみ有効です。
11. Open Event Generator - イベントジェネレータを開く
    このボタンは、CT ブロックまたはハイブリッドプロジェクトの実験を行っている場合は無効です。
12. Open Event Tracer - イベントトレーサを開く（データのトレースを行うための“Event Tracer”ウィンドウが開きます）
13. Open Data Generator - データジェネレータを開く
14. Open Data Logger - データロガーを開く
15. Update Dependent Parameters - 依存パラメータの更新
16. Expand / Collapse Window - コンポーネントの表示／非表示
17. Always on top - 常に前面に表示（“Physical Experiment”ウィンドウが常にデスクトップの最前面に表示されるようにします）
18. Navigate down to child component - チャイルドコンポーネント（下位のコンポーネント）に移動
19. Navigate up to parent component - ペアレントコンポーネント（上位のコンポーネント）に移動

5.3 キーボードでの操作

ASCET は、キーボードを使ってより迅速で正確な操作を行うことができます。使用されているキーには、ユーザーの一般的な傾向、つまり **<Ctrl>** や **<Alt>** との組合せを使用するキーボードショートカットよりもファンクションキー **<F1>** ～ **<F12>** の方が好まれ、さらに単独のキーを使用する方がより好ましい、という条件が取り入れられています。**<Ctrl> + <F1>** を押すと、現在有効なキーボードコマンドの一覧が表示されます。

5.3.1 一般的なキーボード操作

以下のテーブルには、ASCET の実験環境を使用する際に特に重要なキーボードコマンドがまとめられています。その他すべてのキーボードコマンドについては、213 ページの「キーボード操作」の章を参照してください。

キー	機能
<F2>	入力モードに切り替える（テーブルの編集時など）
<Shift> + <F10>	選択されたアイテムのショートカットメニューを開く
<Alt>	メインメニューをアクティブにする
<Alt> + <F4>	アクティブなウィンドウを閉じる（コンポーネントマネージャの場合は ASCET を終了する）
<Alt> + <F6>	現在開いている他の ASCET ウィンドウに切り替える
<Alt> + <Space>	アプリケーションウィンドウのシステムメニューを開く
<Alt> + <Tab>	現在開いている他のアプリケーションに切り替える
<Ctrl> + <F1>	ホットキー（キーボードコマンド）の一覧を表示する
<Ctrl> + <a>	全アイテムを選択する（例：リスト内の全アイテム）
<Ctrl> + <c>	データをクリップボードにコピーする
<Ctrl> + <v>	クリップボードのデータを貼り付ける
<Ctrl> + <x>	データを切り取ってクリップボードに貼り付ける
<Ctrl> + <y>	最後の操作を取り消す（エディタでのみ有効）
<Ctrl> + <z>	最後の操作を繰り返す（エディタでのみ有効）
< ↓ >, < ↑ >, < ← >, < → >	テーブルまたはリスト内のアイテムを選択するカーソルを移動（また階層ビューにおいては < → > でフォルダが開き、< ← > でフォルダが閉じます）
<Enter>	入力内容を確定して入力モードを終了する（また階層ビューにおいては < → >< ← > キーと同様の機能を持ちます）
<Del>	選択されたアイテムの削除
<Esc>	入力内容を取り消して入力モードを終了する
<Space>	テーブルまたはリスト内のアイテムの選択／解除
<Tab>	フォーカス（強調表示）を、ウィンドウ内の次のアイテムまたはオプションに移動（<Shift> + <Tab> で逆方向に移動します）

5.3.2 Windows の規則に基くキーボード制御

各種メニューの巡回やウィンドウの起動など、ASCET の一般的な操作方法は、Windows® の基本規則に準じています。

<Alt> キーを押したまま、メニュー内にアンダーライン付きで示された文字のキーを押すと、対応するコマンドが起動されます。各サブメニューコマンドを起動するには、**<Shift>** キーを押したままアンダーライン付きの文字のキーを押します。

たとえばキーボードで **File** メニューを開くには、**<Alt> + <F>** を押してください。

現在作業を行っているウィンドウ内で **<Tab>** キーを押すと、フォーカスが次のアイテムに順に（左上から右下へ）切り替わります。また、**<Alt>** キーを押したまま各フィールドのタイトル内のアンダーラインのついた文字を押すと、そのフィールドがフォーカス（選択）されます。

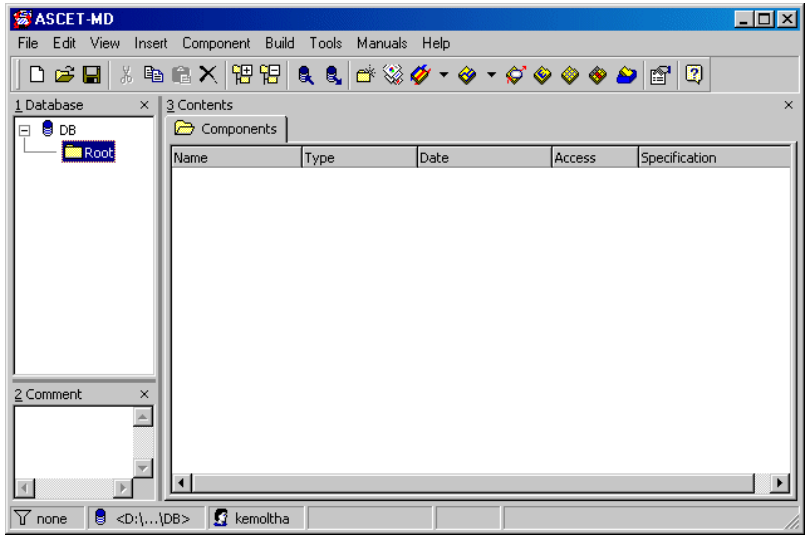
ウィンドウ内で、他の機能のために **<Tab>** が使用されている場合、たとえばテキスト編集時などは、キーボードショートカット **<Ctrl> + <Tab>** を使用すれば、次のウィンドウエレメントに切り替えることができます。（左上から右下の順に）

リストボックス内で矢印キーを押すと、次のアイテムに切り替わり、**<Shift>** キーを押しながら同じ操作を行えば、複数のアイテムを選択できます。

Windows の規則に従い、**<Alt> + <Tab>** を押すと、アクティブなアプリケーションが切り替わります。この際、ASCET のサブシステム（コンポーネントエディタ、実験環境、インプリメンテーションエディタ、データエディタ等）は、独立したアプリケーションとして扱われます。

ウィンドウ内のタブやフィールド（たとえばコンポーネントマネージャの“3 Components”フィールド）の切り替えは、Windows の規則に従い、**<Ctrl> + <Tab>** で行います。

ウィンドウ内で、**<Alt>** キーを押したまま、フィールドやリストのタイトルに示されたアンダーライン付きの数字を押すと、そのフィールドやリストにフォーカスを移動することができます。たとえば以下のウィンドウで **<Alt> + <2>** を押すと、“2 Comment”テキストフィールドがアクティブになります。



5.4 マウスでの操作

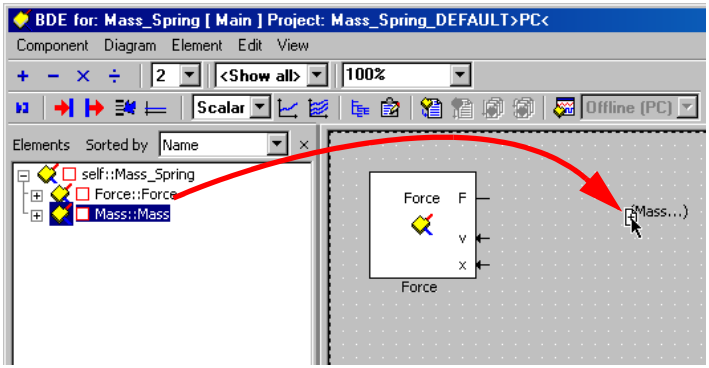
オフィスや実験室では、マウスを使用して ASCET をより快適に操作できます。マウスの使用法は、Windows® の基本規則に準じています。

複数のアイテムを選択するには、**<Shift>** または **<Ctrl>** キーを使用します。

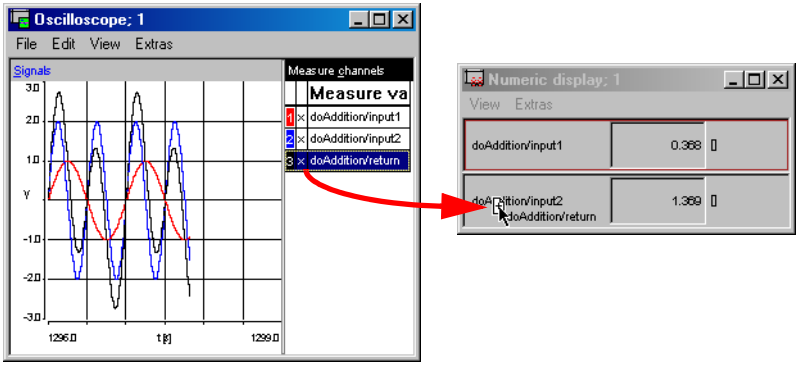
ウィンドウ内のエレメント上にポインタを合わせてマウスの右ボタンをクリックすると、そのアイテムに応じたショートカットメニューが開きます。

5.4.1 ドラッグ & ドロップ

ブロックダイアグラムの作成や測定／適合ウィンドウの設定を行う際、ドラッグ & ドロップ操作で変数を選択できます。使用したい変数にポインタを合わせてマウスの左ボタンをクリックし、ボタンを押し下げたままその変数を目的のウィンドウまたはフィールドまでドラッグします。



変数を他のウィンドウにコピーする場合も、同様の操作で簡単に行えます。ただし、実行不可能な操作を行った場合、たとえばパラメータ（適合変数）を測定ウィンドウにドラッグしようとした場合、その操作は無視されます。



5.5 階層ツリー

ASCET では、データベース内のアイテムなど、階層的なデータ構造を表示する際に階層ビューが使用されます。このツリーに含まれるすべての枝とアイテムを見るには、それぞれの枝を展開したり省略したりする必要があります。また、必要に応じてすべての枝を一度に展開することも可能です。

すべての枝を展開／省略する方法：

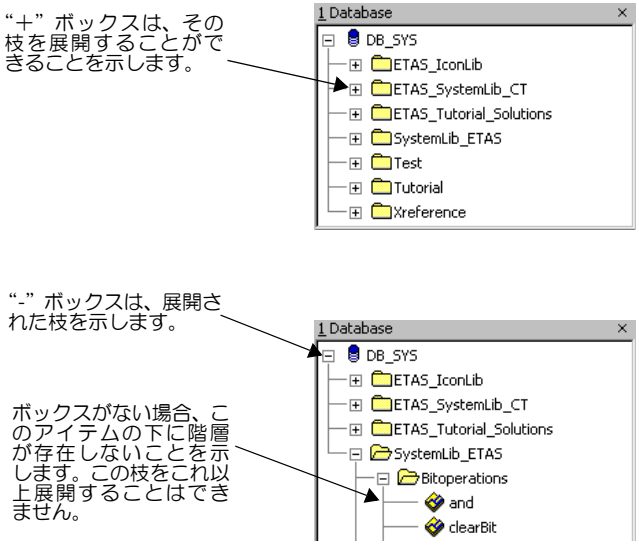
- メニューコマンド **View → Expand All** または **View → Expand All** を実行するか、またはツールバーのボタンを使用します（82 ページの「コンポーネントマネージャのボタン」を参照してください）。

個々の枝を展開／省略する方法：

- 展開したいアイテムの左側の“+”ボックスをクリックするか、または **<+>** キーを押すと、そのアイテムの 1 レベル下の階層が展開表示されます。もう一度同じ“+”ボックスをクリックするか、または **<->** キーを押すと、その階層が閉じます。

  または

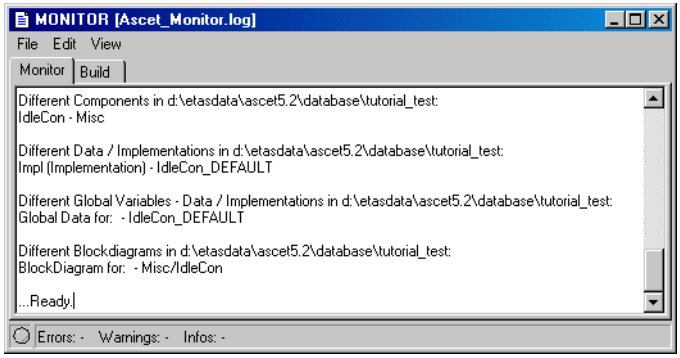
- **< ↑ >**／**< ↓ >**キーでアイテムを選択して**<→>**キーを押すと、そのアイテムが展開表示され、**< ← >** キーを押すとその枝が閉じます。



5.6 ユーザーサポート機能

5.6.1 モニタウィンドウ

ASCET によって実行された処理に関する履歴は、モニタウィンドウ（詳しくは『ASCET ユーザーズガイド』の 2.4 章を参照してください）に出力されます。履歴データには、エラーや通知メッセージなども含まれます。何らかの情報が出力されると、直ちにモニタウィンドウが最前面に表示されます。



モニタウィンドウは、情報を表示するだけでなく、エディタ機能も持っています。

- モニタウィンドウの“Monitor”タブに出力された履歴データは、自由に編集できます。これによって、ユーザーのメモやコメントを ASCET の出力メッセージに付加できます。
- ASCET のメッセージを、付加したコメントと共にテキストファイルとして保存できます。
- 既に保存されている他の履歴データをロードして、記録された処理値内容を比較できます。

5.6.2 キーボードコマンドの一覧

<Ctrl> + <F1> を押すと、現在有効なキーボードコマンドの概要が表示されます。

5.6.3 マニュアルとオンラインヘルプ

デフォルト設定で ASCET をインストールすると、ASCET のユーザーマニュアルがインストールされます。

`ETAS¥ETASManuals` ディレクトリに PDF マニュアル（`ASCET V5.2 Quickstart.pdf`, `ASCET V5.2 Manual.pdf`, `ASCET V5.2 Reference.pdf`）がインストールされ、これらのマニュアルは **Manual** メニューのコマンドによって開くことができます。なお、日本語マニュアルは、**Manual → Open manuals folder** でマニュアルフォルダを開き、そこから直接ファイルを開いてください。

これらの PDF ファイルでは、索引やテキスト検索、または随所に設定されているハイパーリンクを使用して、必要な情報に素早くアクセスすることができます。

また、**<F1>** キーを押すと、ETAS¥ASCET5.2¥Help ディレクトリにインストールされているオンラインヘルプが開きます。

6 チュートリアル

このチュートリアルは、主に、ASCET を初めて使うユーザーを対象として、実際のモデル記述作業を例に ASCET の基本的な使い方を説明するものです。チュートリアルは、互いに関連しあうコンポーネントを扱う複数のレッスンに分かれています。なお、チュートリアルを始める前に、あらかじめ「ASCET を理解する」（49 ページ）を読んで ASCET の基本概念を理解しておいてください。

6.1 単純なブロックダイアグラムを作成する

ASCET では、アプリケーションを構成するメインブロックとしてクラスやモジュール等のコンポーネントを使用します。製品に含まれている既成のコンポーネントを使ったり、独自のコンポーネントを新たに作成することもできます。

ASCET では、コンポーネントは、通常グラフィックを使用して記述します。すべてのコンポーネントの定義が終わったらそれらを組み合わせ、ASCET ソフトウェアシステムの基礎となるプロジェクトを構築します。最終的なソフトウェアシステムは、グラフィカルなモデル記述から生成された C コードで構成され、マイクロコントローラや実験ターゲットコンピュータ上で実行することができます。

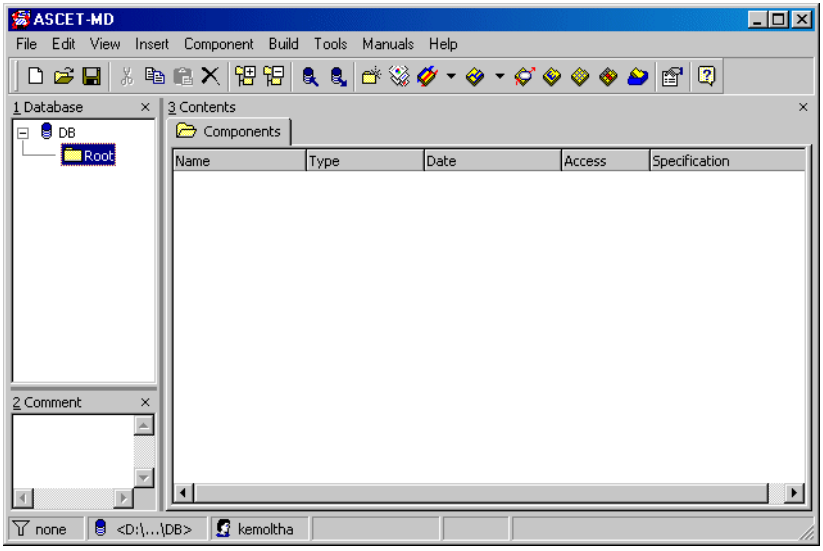
6.1.1 準備

最初に、まずチュートリアルのデータベースを開きます。このチュートリアルで作業するコンポーネントはすべてこのデータベースに格納されるので、すべての作業はこのデータベースを開いた状態で行います。

このチュートリアルで作成するコンポーネントとプロジェクトは、データベース `tutorial` 内の `ETAS_Tutorial_Solutions` というフォルダ内にあらかじめ格納されているので、ここで説明するコンポーネントをすべてユーザー自分で定義する必要はありません。

ただし、少なくともレッスン 1、3、および 4 のコンポーネントだけは、練習のためにご自分で作成することをお勧めします。

ASCET を起動すると、以下のようなコンポーネントマネージャが開き、前回のセッションで開いていたデータベースがロードされます。



`tutorial` データベースがない場合は、以下のようにして新しいデータベースを作成します。

新しいデータベースを作成する：

- コンポーネントマネージャから、**File → New Database** を選択します。

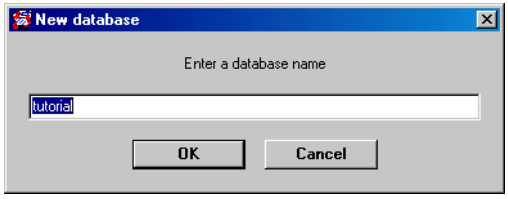
または

- **New** ボタンをクリックします。

または

- **<Ctrl> + <N>** を押します。

“New database”ダイアログボックスが開きます。



- `Root` という名前を入力します。

- **OK** をクリックします。

  新しいデータベースが作成されます。ここには、データベース名と `Default` というフォルダのみが含まれます。

`tutorial` データベースすでに存在している場合は、以下のようにしてそのデータベースを開きます。

データベースを開く：

- コンポーネントマネージャから、**File → Open Database** を選択します。

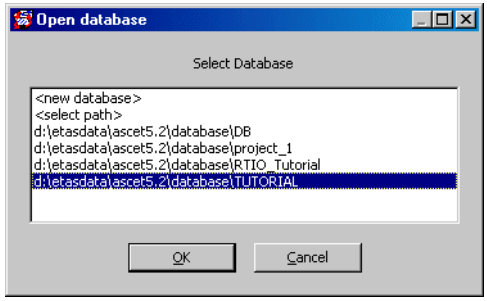
または

- **Open** ボタンをクリックします。

または

- **<Ctrl> + <O>** を押します。

  “Open database”ダイアログボックスが開き、現在設定されているデータベースパスに存在するデータベースの一覧が表示されます。



- 一覧の中に `tutorial` データベースがある場合は、それを選択して **OK** をクリックします。

  コンポーネントマネージャに `tutorial` データベースの内容が表示されます。

- `tutorial` データベースが他のパスに格納されている場合には、`<select path>` オプションを選択して **OK** をクリックします。
- “Select database path”ダイアログボックスが開くので、データベースを指定して **OK** をクリックします。

新しいコンポーネントを作成するために、まず `Tutorial` という名前の新しいトップレベルのフォルダと `LessonN` という各レッスン用のサブフォルダを作成します。

新しいトップレベルフォルダを作成する：

- “1 Database”ペーンでデータベース名を選択します。
- メニューアイテム **Insert** → **Folder** を選択します。

または

- **Insert Folder** ボタンをクリックします。

または

- **<Insert>** を押します。

  “1 Database”ペーンに `Root` という名前の新しいトップレベルのフォルダが表示されます。
- トップレベルフォルダの名前を`Tutorial`に変更します。強調表示されている名前に上書き入力してから **<Enter>** を押します。
- フォルダ `Tutorial` を選択します。
- もう一度 **Insert** → **Folder** を選択します。

  “1 Database”ペーンに `Folder` という名前の新しいフォルダが表示されます。
- 新しいフォルダの名前を `Lesson1` に変更します。

このチュートリアルで作成するコンポーネントはすべて `LessonN` フォルダに格納します。各レッスンごとに新しいフォルダを作成してください。データベースにはトップレベルフォルダが必ず 1 つ以上あり、その中に任意の数のサブフォルダを作成することができます。

注記

ASCET 5.x では、すべてのフォルダ名やアイテム名、およびそれらの中の変数やメソッドの名前は、ANSI C 標準に準拠している必要があります。

次に、最初のコンポーネントを `Lesson1` フォルダに作成します。

コンポーネントを作成する：

- “1 Database”ペーンのフォルダ `Lesson1` をクリックします。
- **Insert** → **Class** → **Block Diagram** を選択します。

  “1 Database”ペーンの `Lesson1` フォルダの下に、`Class_Blockdiagram` という新しいコンポーネントが表示されます。このコンポーネントは *class* タイプです。class タイプは ASCET で頻繁に使用されます。
- このコンポーネントの名前を`Addition`に変更します。

6.1.2 クラスを定義する

`Tutorial/Lesson1` フォルダに新しいコンポーネントを作成したので、次にその機能を定義します。まず最初にコンポーネントのインターフェース、つまり、メソッド、引数、および戻り値を定義し、次にそのコンポーネントの機能を定義するブロックダイアグラムを作成します。

コンポーネントの機能を定義する：

- “1 Database”ペーンでコンポーネント `Addition` を選択します。
- メニューアイテム **Component → Edit Item** を選択します。

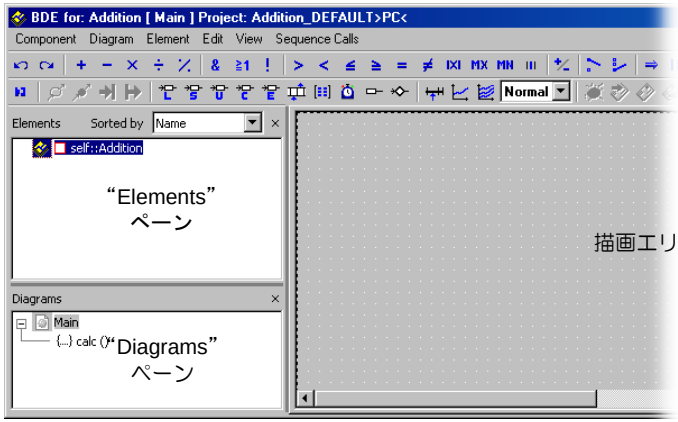
または

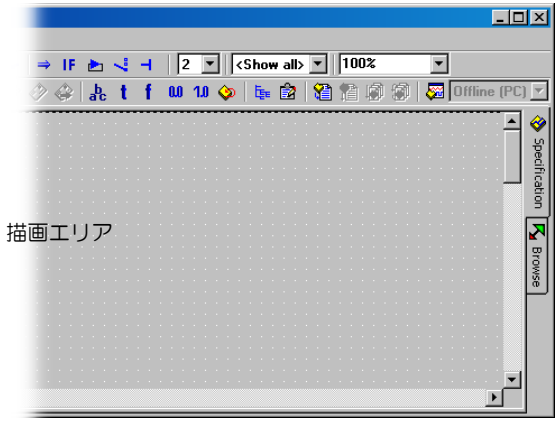
- コンポーネントをダブルクリックします。

または

- **<Enter>** を押します。

  ブロックダイアグラムエディタが開きます。これは、コンポーネントの機能を定義するためのメインウィンドウです。





- “Diagrams”ペーンに表示されているメソッド `calc` を選択します。このメソッドは、デフォルトメソッドとして自動的に作成されたものです。
- **Diagram → Rename** を選択します。

または

- **<F2>** を押します。

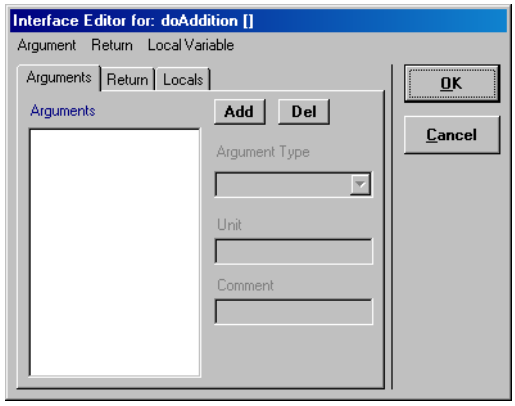
  メソッド名 `calc` が強調表示されます。
- このメソッドの名前を `doAddition` に変更します。

- **Diagram → Edit** を選択します。

または

- メソッド名をダブルクリックします。

  このメソッドのインターフェースエディタが開きます。



クラスには、少なくとも 1 つのメソッドを定義する必要があります。ASCET におけるメソッドは、オブジェクト指向プログラミング言語で用いられる「メソッド」や、手続き型プログラミング言語で用いられる「関数」に似ています。メソッドは任意の数の引数を入力し、1 つの戻り値を出力することができます（これらはすべて任意指定です）。引数は、コンポーネントにデータを渡すために使用され、戻り値はコンポーネント内で行われた計算の結果を「外側」に返すために使用されます。

これらの「メソッドインターフェース」は、インターフェースエディタを使用して定義します。ここでは `continuous` 型の引数を 2 つと戻り値を 1 つ定義します。

メソッドインターフェースを定義する：

- インターフェースエディタで **Argument → Add** を選択します。

  “Arguments” ペーンに、`arg` という新しい引数が表示されます。

- この引数の名前を `input1` にします。
- もう 1 つの引数を作成し、`input2` という名前を付けます。

  デフォルトでは引数の型は *continuous*（または略して *cont* ）になっています。ここでは、この型のまま使用します。

- インターフェースエディタの "Return" タブを選択します。
- **Return Value**チェックボックスにチェックマークを付けます。

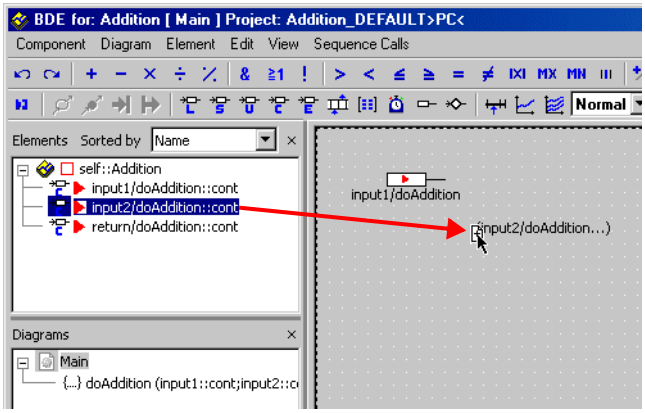
  戻り値の型も、デフォルトで *cont* になります。
- **OK** をクリックして、インターフェースエディタを閉じます。

メソッド `doAddition` の引数と戻り値の名前が、ブロックダイアグラムエディタの左側の“Elements”ペーンに表示されます。これで、ブロックダイアグラムを作成してこのコンポーネントの機能を定義する準備が整いました。

コンポーネント Addition の機能を定義する：

- “Elements”ペーンの第 1 引数をドラッグして、ブロックダイアグラムエディタの描画エリアにドロップします。

  この引数のシンボルが、描画エリアに表示されます。



- 同じドラッグアンドドロップ操作により、他の引数と戻り値を追加します。

- **Addition** ボタンをクリックします。

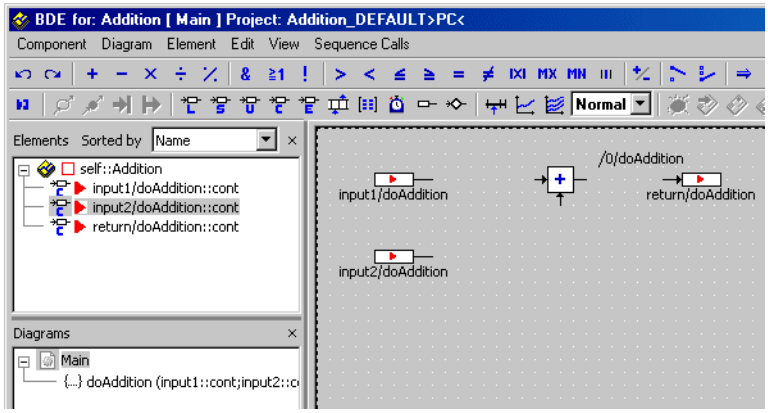
  マウスカーソルに加算演算子がロードされました。

- 描画エリア内の、引数のシンボルと戻り値のシンボルの間をクリックします。

  加算シンボルが挿入されます。デフォルトでは、これには 2 つの入力ピン（矢印で示されます）と、1 つの出力ピンが付いています。出力ピンは右側に付きます。

エレメントと演算子は、描画エリアの任意の位置にドラッグして配置することができます。

次に、エレメント同士を接続して、情報の流れを定義します。



ダイアグラムエレメントを接続する：

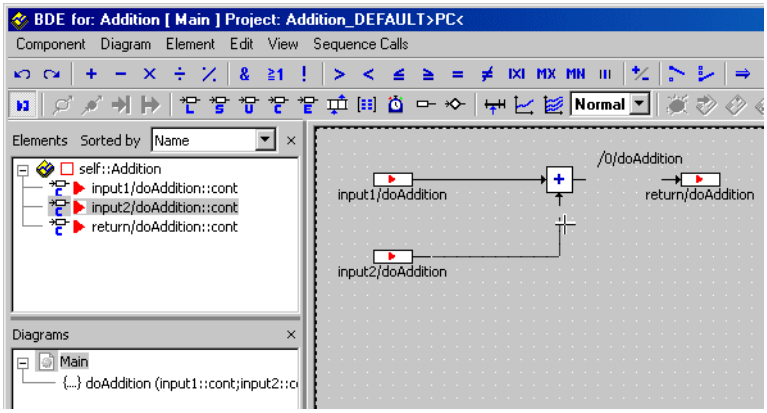
- **Connect** ボタンをクリックします。
- または、描画エリア内の、エレメントが配置されていない部分を右クリックします。

  カーソルは、描画エリア内で十字カーソルに変わります。
- 第 1 引数を表すシンボルの出力ピンをクリックして、接続を開始します。

  マウスカーソルを動かすと、それに従って線が引かれます。描画エリア内でクリックするたびに、その時点で引かれていた線が固定されます。このようにして、接続線の道筋を確定していきます。

- 加算シンボルの左側にある入力をクリックします。

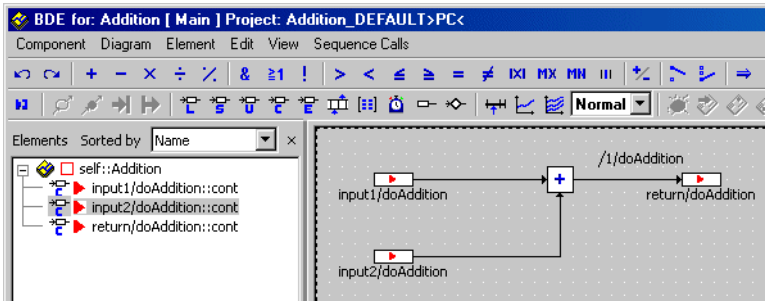
  これで、第 1 引数のシンボルが加算シンボルの入力に接続されます。



- 第 2 引数のシンボルを、加算シンボルのもう 1 つの入力に接続します。
- 戻り値のシンボルを、加算シンボルの出力に接続します。

  加算演算子と戻り値との接続が、緑色の線で表示されます。緑色の線は、この演算のシーケンスがまだ決定されていないときうことを表わしています。
- **Sequence Calls → Sequencing - Ignore Info** を選択して、加算のシーケンスを自動的に決定します。

  加算演算子と戻り値の間の接続が黒色の線で表示されます。



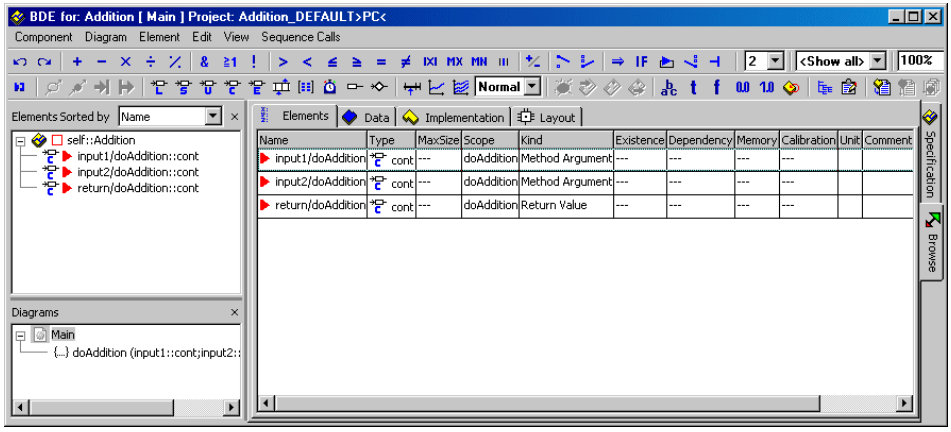
これで、最初のコンポーネントの定義、つまり機能記述は完了です。

最後に、コンポーネントのレイアウト、つまり、このコンポーネントが他のコンポーネント内で使用される際にどのように表示されるようにするかを定義します。

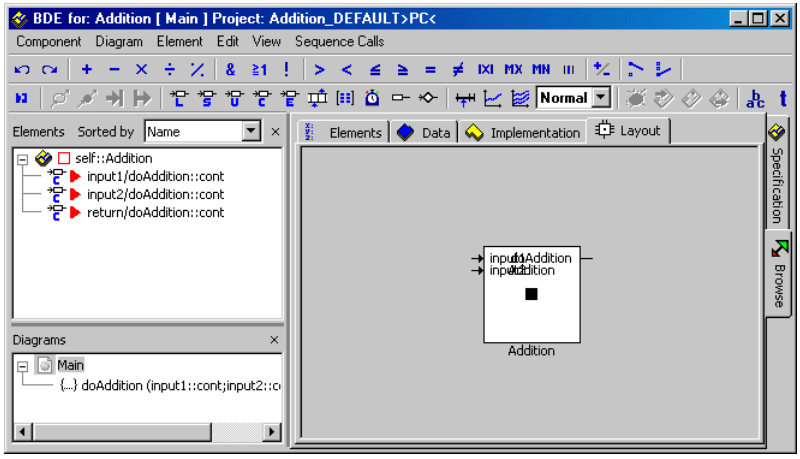
コンポーネントのレイアウトを編集する：

レイアウトの編集方法は 2 通りあります。

- ウィンドウ右端の“Browse”タブを選択して“Information/Browse”ビューに切り替えます。

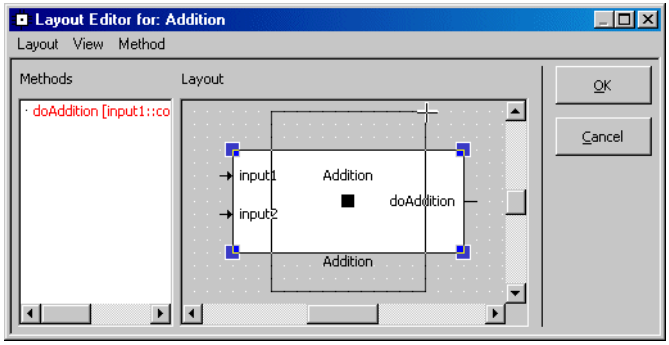


- “Layout”タブを選択し、レイアウトが表示されたエリアをダブルクリックしてレイアウトエディタを開きます。



- または、**Component** → **Edit Layout** を選択します。

  レイアウトエディタ（“Layout Editor”ダイアログ）が開きます。



- ブロックをクリックするとそのブロックのハンドルが表示されるので、そのハンドルをドラッグしてブロックを任意のサイズに変更します。
- 引数と戻り値のピンをドラッグして、配置を調整します。
- **OK** をクリックします。

これで、コンポーネントは完成しました。このレッスンの最後の作業として、コンポーネントをデータベースに保存します。

コンポーネント `Addition` を保存する：

- **Diagram** → **Store to Cache** を選択し、ブロックダイアグラムエディタを閉じます。
- コンポーネントマネージャから **File** → **Save Database** を選択します。

または

- **Save** ボタンをクリックします。

  作成したコンポーネントは、この操作をするまではディスクに書き込まれません。**Store to Cache** を選択した場合、変更部分はキャッシュメモリに格納されるだけです。そこで、作業中には定期的に保存を行うことをお勧めします。

  ユーザーオプションを設定して、変更内容が自動的に保存されるようにすることもできます（『ASCET ユーザーズガイドの「一般オプションの設定」という項を参照してください）。

ここで、練習のため、同じ機能を ESDL（**E**mbedded **S**oftware **D**escription **L**anguage）でモデリングしてみましょう。そうすれば、ESDL エディタや、外部ソースコードエディタの使用法を習得することができます。

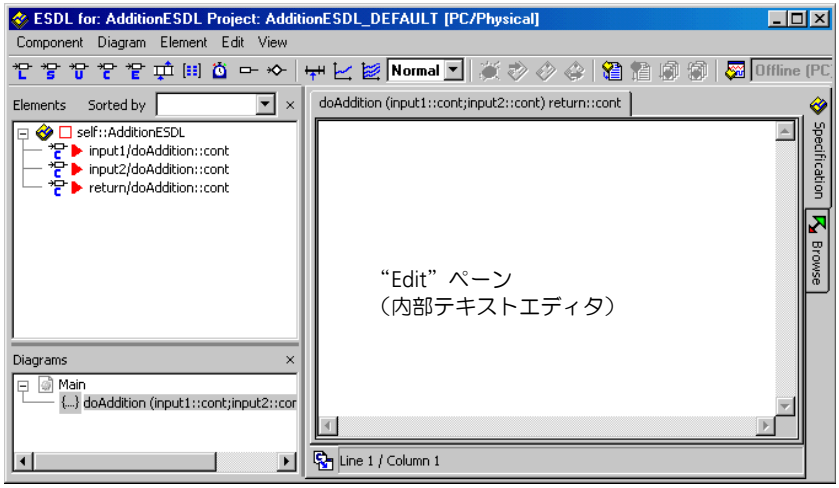
まず初めに、モジュールインターフェースを ESDL タイプの新しいモジュールにコピーしてから、そのモジュールの名前を変更します。それから、そのモジュールで実現したい機能を、ASCET の ESDL エディタ、または外部テキストエディタを使用して記述します。

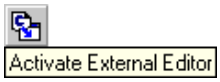
コンポーネント（インターフェース）`Addition` をコピーして定義する：

- コンポーネントマネージャの“1 Database”ペーンでコンポーネント `Addition` を選択します。
- メニューから **Component → Reproduce As → ESDL** を選択します。

  コンポーネントのコピーが作成され、`Addition1` という名前が自動的に付けられます。
- 新しいコンポーネントの名前を `AdditionESDL` に変更します。
- “1 Database”ペーンで、新しいコンポーネントの名前をダブルクリックします。

  `AdditionESDL` 用のエディタが開きます。このエディタにもさまざまな編集機能が備わっています。





- 内部テキストエディタの“Edit”ペーンに、以下のような機能を入力します。

  ```
  return input1 + input2;
  ```

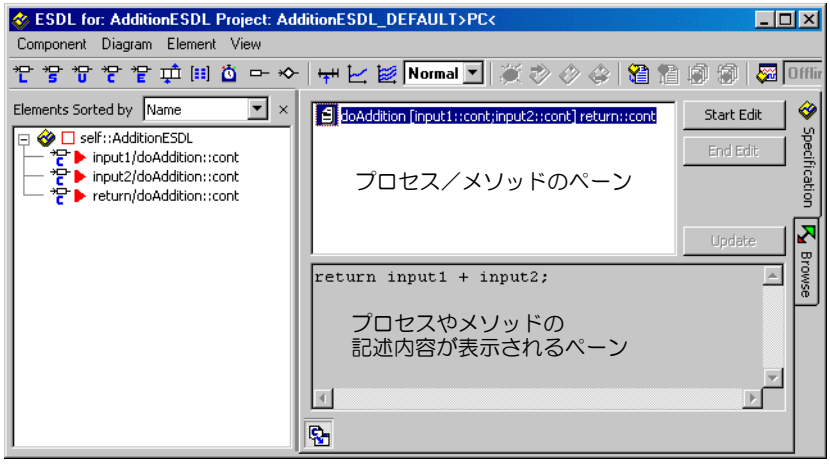
- 今度は **Activate External Editor** ボタンをクリックして、外部エディタモードに切り替えます。

  変更内容を保存するかどうか、尋ねられます。

- **Yes** をクリックして確定します。

  変更内容が保存され、ESDL エディタは外部エディタモードに切り替わります。

  外部エディタモードに切り替わると、以下のような画面になります。



- プロセス／メソッドのペーンから、定義したいメソッドまたはプロセスを選択します。

  すでに入力されている機能が定義フィールドに表示され、**Start Edit** ボタンが有効になります。

- **Start Edit** をクリックして外部エディタを起動します。

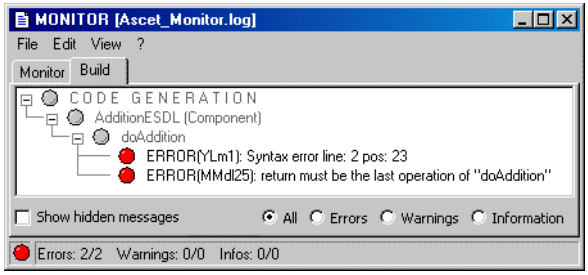
注記

外部エディタが起動されると、Windows に登録されている、*.c と *.h というファイルに割り当てられたアプリケーションが呼び出されます。この外部エディタと ASCET 間のデータ転送は一時ファイルを介して行われるので、外部エディタを閉じる前や ESDL エディタにデータを転送する前には、ファイルが保存されていることを確認してください。

Start Edit ボタンが無効になり、**End Edit** および **Update** ボタンが有効になります。

- 外部エディタで機能を記述します。
- 記述した機能を保存します。
- ESDL エディタで **Update** をクリックし、外部エディタからデータを転送します。
- 最後に **End Edit** をクリックして、外部エディタとの連結を解除します。

  End Edit をクリックしても外部エディタは開いたままですが、それ以降、外部エディタで変更した内容を ESDL エディタに転送することはできません。
- 再度 **Activate External Editor** をクリックして、外部エディタモードを終了します。
- **Diagram → Analyze Diagram** を選択して、入力したコードを検証します。エラーがあった場合は、ASCET モニタウィンドウに表示されます。



6.1.3 まとめ

このレッスンでは、ASCET で以下の作業を行いました。

- データベースを開く

- フォルダを作成して名前を付ける
- コンポーネントを作成して名前を付ける
- メソッドのインターフェースを定義する
- 描画エリアにダイアグラムエレメントを配置する
- ダイアグラムエレメントを接続する
- コンポーネントのレイアウトを編集する
- 機能記述用のビュー（“Specification”ビュー）とブラウザビュー（“Browse”ビュー）とを切り替える
- コンポーネントを保存する
- コンポーネントのインターフェースをコピーする
- ESDL エディタを使用する
- 外部エディタを使用する

6.2 コンポーネントの実験を行う

`Addition` および `AdditionESDL` というコンポーネントができあがったので、それを用いて実験を行います。実験環境では、コンポーネントがどのように機能するかを、シミュレーションによって運用時と同様に見ることができます。実験環境には各種ツールが用意されていて、コンポーネント内の入力、出力、パラメータ（適合変数）、および変数（測定変数）の値をモニタすることが可能です。

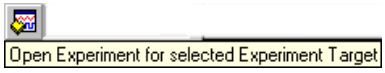
6.2.1 実験環境（Experiment Environment）を起動する

実験環境を、ブロックダイアグラムエディタまたは ESDL エディタから起動します。起動するには、実験したいコンポーネントを指定して実験環境を開きます。

実験環境を起動する：

- ASCET コンポーネントマネージャから、`¥Tutorial¥Lesson1` フォルダに入っている `Addition` のブロックダイアグラムを開きます。
- ブロックダイアグラムエディタで **Component → Open Experiment** を選択します。

または

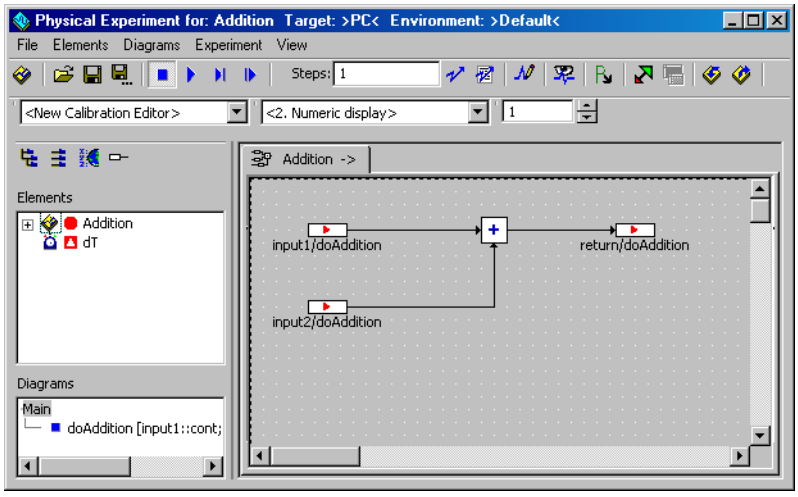


- **Open Experiment for selected Experiment Target** ボタンをクリックします。

  実験用のコード生成が行われます。ASCET はユーザーが定義したモデルを分析し、そのモデルで記述されている機能を実現する C コードを生成します。さまざまなプラットフォーム用のコードを生成することができます。

このレッスンでは、デフォルト設定を使用して PC 用のコードを生成します。

コードの生成とコンパイルが終わると、実験環境が開きます。



6.2.2 実験をセットアップする

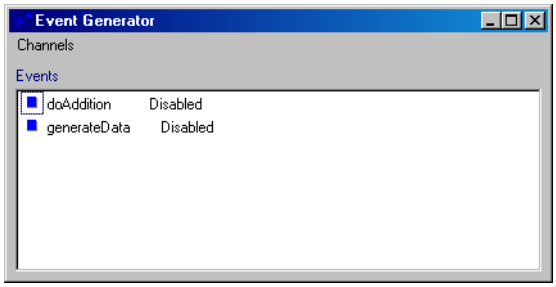
実際に実験を開始する前に、環境の設定が必要です。つまり、実験用に生成する入力値を指定し、さらに実験結果をどのように表示するかを指定します。このために、イベントジェネレータ、データジェネレータ、そして最後に測定システムを順にセットアップしていきます。

イベントジェネレータをセットアップする：

- **Open Event Generator** ボタンをクリックします。

  “Event Generator”ウィンドウが開きます。シミュレートする各メソッドごとに、イベントを 1 つと `generateData` イベントを作成する必要が

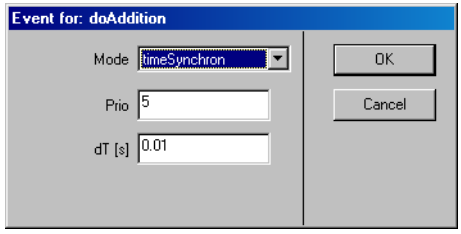
あります。運用時にはオペレーティングシステムが行うスケジューリングが、これらのイベントによってシミュレートされます。



- `doAddition` イベントを選択します。
- **Channels → Enable** を選択します。
- もう一度、イベント `doAddition` を選択します。
- **Channels → Edit** を選択します。

  これら 2 つの操作は、"Elements" フィールドのショートカットメニューからも行えます。

  “Event”ダイアログボックスが開きます。

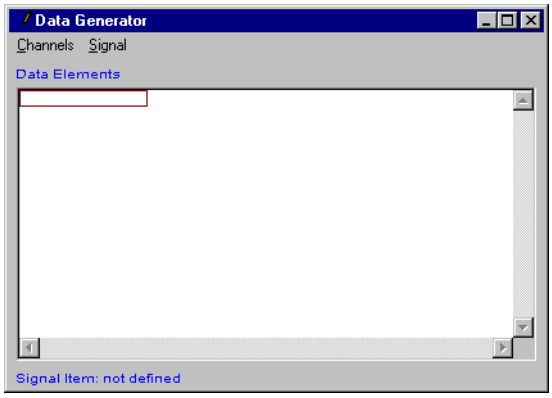


- `dT` の値を 0.001 にします。
- **OK** をクリックします。
- イベントジェネレータ内で `generateData` イベントを選択し、その `dT` 値を 0.001 にします。
- “Event Generator”ウィンドウを閉じます。

データジェネレータをセットアップする：

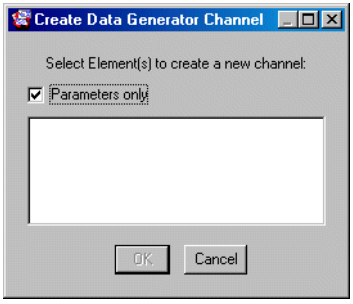
- **Open Data Generator** ボタンをクリックします。

  “Data Generator”ウィンドウが開きます。



- **Channels → Create** を選択します。

  “Create Data Generator Channel”ダイアログボックスが開きます。

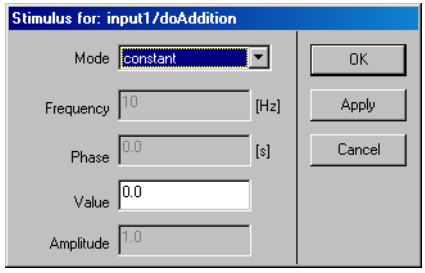


- 必要に応じて **Parameters only** オプションをオフにして、パラメータと変数の両方が一覧に表示されるようにします。
- 一覧から、`input1/doAddition` および `input2/doAddition` という変数を選択します。
- **OK** をクリックします。

  2 つの変数が“Data Generator”ウィンドウの“Data Elements”ペーンにリストアップされます。

- “Data Elements”ペーンの `input1/doAddition` を選択します。
- **Channels → Edit** を選択します。

  “Stimulus”ダイアログボックスが開きます。



- 以下の値を設定します。

  Mode： sinus

  Frequency： 1.0Hz

  Phase： 0.0s

  Offset： 0.0

  Amplitude： 1.0

- **OK** をクリックして“Stimulus”ダイアログボックスを閉じます。
- 同様にして `input2` の値を以下のように設定します。

  Mode： sinus

  Frequency： 2.0Hz

  Phase： 0.0s

  Offset： 0.0

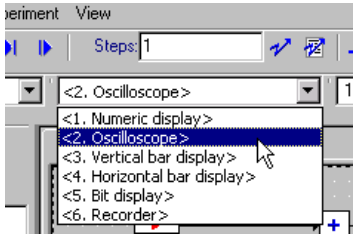
  Amplitude： 2.0

- “Data Generator”ウィンドウを閉じます。

上記のように設定すると、周波数と振幅が異なる 2 つの正弦波が得られます。`Addition` コンポーネントはこの 2 つの正弦波を合成し、その結果として得られるカーブを出力します。

これらのカーブをオシロスコープに表示して波形を確認できるようにするため、測定システムをセットアップします。

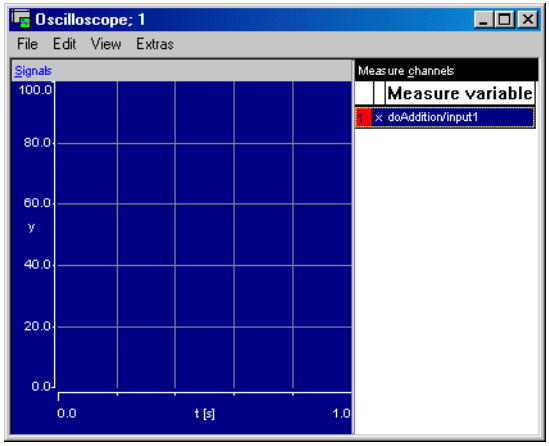
- “Physical Experiment”ウィンドウの“Measure View”コンボボックスで、データ表示タイプとして <2. Oscilloscope> を選択します。



- “Elements”ペーン内で、Addition の隣の“＋”シンボルをクリックして、エレメントリストを開きます。

  Addition コンポーネントのエレメントが表示されます。
- input1/doAddition を選択します。
- **Elements → Measure** を選択します。

  input1 という測定チャンネルが割り当てられたオシロスコープウィンドウが開きます。実験環境の“Measure View”リストが更新され、そこにこの測定ウィンドウのタイトルが追加されます。



- “Physical Experiment”ウィンドウの“Elements”リスト内の input2/doAddition を選択します。

- **Elements → Measure** を選択します。

  `input2` という測定チャンネルがオシロスコープに追加されます。
- 同様にして `return/doAddition` を測定ウィンドウに追加します。
- 実験環境で、**File → Save Environment** を選択します。

これで実験のセットアップが終わり、実験を開始することができるようになりました。環境設定を保存したので、次回、このコンポーネント用の実験環境を起動すると、同じ環境が再びロードされます。

6.2.3 実験環境を使用する

実験環境には、コンポーネント内の変数の値を表示したり、実験実行中にセットアップ内容を変更するための機能が用意されています。また、値の表示方法を選択したり調整することもできます。

実験を開始する：

- “Physical Experiment”ウィンドウの **Start Offline Experiment** ボタンをクリックして実験を開始します。

  シミュレーションが実行され、その結果がオシロスコープに表示されます。

- **Stop Offline Experiment**ボタンをクリックして、実験を停止します。

この時点では、オシロスコープには、カーブのごく一部しか表示されていません。オシロスコープにカーブの他の部分を表示するには、値軸（信号値を示す Y 座標軸）のスケールを変更する必要があります。

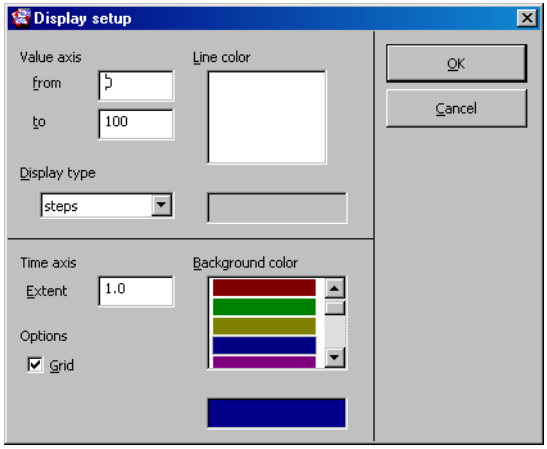
オシロスコープのスケールを変更する：

- オシロスコープウィンドウの“Measure Channels”リストから、3 つのチャンネルをすべて選択します。

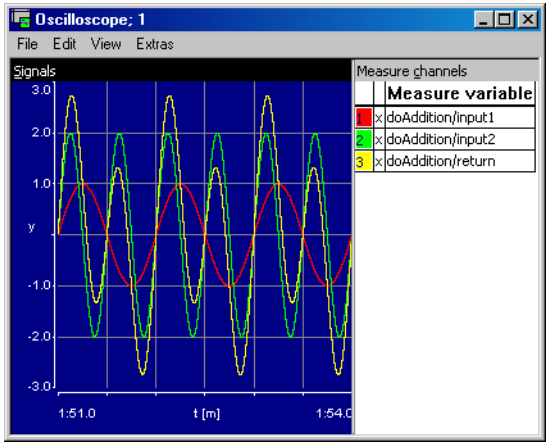
  複数のチャンネルを選択するには、**<Ctrl>** キーを押し下げたまま個々のチャンネルをクリックします。

  すべてのデータエレメントが強調表示され、これから行う変更は 3 つのチャンネルすべてに反映されます。
- **Extras → Setup** を選択します。

“Display Setup”ダイアログボックスが開きます。



- “Value Axis”の範囲を -3 ～ 3 に変更します。
- “Time Axis Extent”を 3 にします。
- “Backgroud color”リストから背景色を選択します。
- **<Enter>** を押します。



これで、値が適切なスケーリングで表示されるようになりました。入力された 2 つの正弦波と、その 2 つを合成した出力波形が表示されます。入力値を調整して、出力がどのように影響を受けるかを調べてみましょう。

実験の入力値を変更する：

- “Physical Experiment”ウィンドウで **Experiment → Data Generator** を選択して、“Data Generator”ウィンドウを開きます。
- データジェネレータ上で、変更したい変数を選択します。
- **Channels → Edit** を選択します。
  “Stimulus”ダイアログボックスが開きます。
- 値を適宜調整します。
- **Apply** をクリックします。

オシロスコープのカーブが、新しい設定に応じて変化します。実験の実行中に、すべての設定を変更することができます。

6.2.4 まとめ

このレッスンでは、ASCET で以下の作業を行いました。

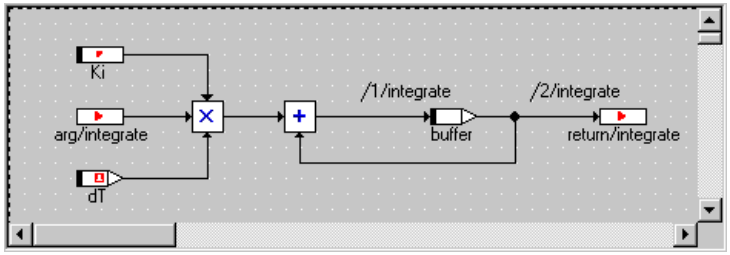
- 実験環境を呼び出す
- イベントジェネレータをセットアップする
- データジェネレータをセットアップする
- 測定システムをセットアップする
- 実験を開始し、停止する
- 実験の環境設定を保存する
- 実験実行中にスティミュレーションの内容を変更する

6.3 再利用可能なコンポーネントを定義する

このレッスンでは、マイクロコントローラソフトウェアでよく用いられる機能単位である「積分器」のクラスを作成します。このダイアグラムはやや複雑ですが、ここまでに学習した方法で作成し、実験することができます。

この例では、与えられる時間と速度から走行距離を算出する積分器を定義します。速度はメートル／秒の単位で与えられ、それを dT（秒）の周期で積算します。各周期ごとに値がアキュムレータに累算され、アキュムレータの値は、所定の時間経過後の走行距離（メートル）となります。

ASCET では、アキュムレータなどの標準ブロックは、シンプルに図示されます。



6.3.1 ダイアグラムを作成する

ダイアグラムを記述する前に、`addition` コンポーネントの場合と同様の準備作業が必要です。まず `Tutorial` フォルダ内に新しいフォルダを作成してから、新しいクラスを追加します。これで、メソッドのインターフェース、さらにブロックダイアグラムとレイアウトを定義することができます。

まず、フォルダと新しいクラスを作成することから始めます。

積分器のクラスを作成する：

- コンポーネントマネージャで、`Tutorial` フォルダを開きます。
- 新しいフォルダを作成し、その名前を `Lesson3` にします。
- `Lesson3` フォルダ内に新しいクラスを作成し、その名前を `Integrator` にします。

積分器のインターフェースを定義する：

- “1 Database”ペーンで、エレメント `Integrator` を選択します。
- エレメントをダブルクリックするか、または **Component → Edit Item** を選択します。

  ブロックダイアグラムエディタが開きます。
- “Diagrams”ペーンのメソッド `calc` の名前を `integrate` に変更します。
- メソッド `integrate` を編集して、1 つの引数（`cont` 型）と戻り値（`cont` 型）を追加します。
- **OK** をクリックします。

  インターフェースエディタが閉じて、ブロックダイアグラムエディタに戻ります。
- `integrate` の引数と戻り値を描画エリアに配置します。

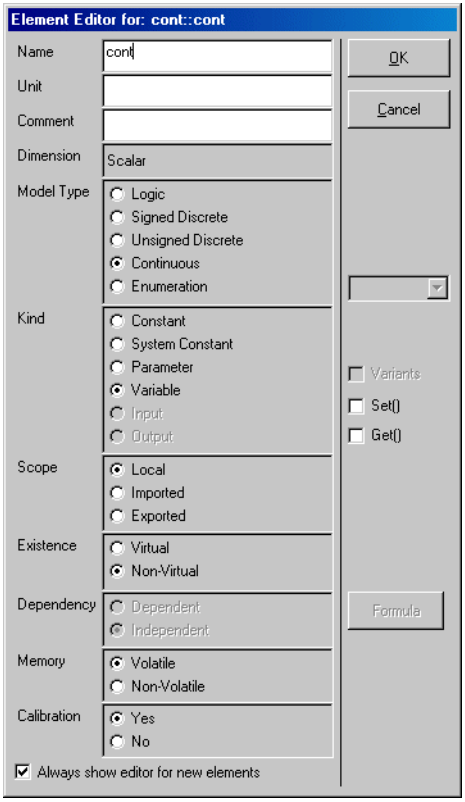
この積分器は、ここまでのレッスンでは使用されなかった「変数」および「パラメータ」という 2 つのタイプのエレメントを使用します。

「変数」は、プログラミング言語の変数と同じように用いられます。変数には値を格納することができ、以降の計算ではその値を読みとることができます。これとは対照的に、「パラメータ」は読みとり専用です。パラメータの値は、たとえば新しい実験環境での適合作業によってなど、コンポーネントの外部でしか変更できず、コンポーネント内での計算で上書きすることはできません。なお ETAS の測定・適合ツールでは、「変数」は「測定変数」、「パラメータ」は「適合変数」と呼ばれています。

さらにこの例では、「依存パラメータ 」を定義します。しかし、これは積分器の機能とは無関係です。依存パラメータ は 1 つまたは複数のパラメータに依存するもので、その値は別のパラメータの値から算出されます。この計算は値が定義された時や適合された時にだけ行われます。依存パラメータは、ターゲットコード内においては通常のパラメータと全く同じように機能します。

変数を作成する：

- **Continuous** ボタンをクリックします。
  エレメントエディタが開きます。



- “Name”フィールドに `buffer` という名前を入力します。
- **OK** をクリックします。
  変数の名前が `buffer` になりました。マウスカーソルにこの continuous 型の変数がロードされ、カーソルが十字カーソルに変わります。
- 描画エリア内をクリックして、変数を配置します。
  変数が描画エリア内に配置され、“Elements”リスト内にその変数の名前が反転表示の状態で追加されます。

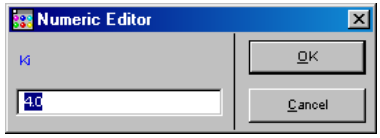
上記の操作でエレメントエディタが自動的に開かない場合は、そのまま変数を描画エリアに配置し、その後、“Elements”リスト内のその変数をダブルクリックしてエレメントエディタを開いてください。エレメントエディタの **Always show dialog for new elements** オプションをオンにしておけば、次回のエレメント作成時から、自動的にエレメントエディタが開くようになります。

パラメータを作成する：

- **Continuous Parameter** ボタンをクリックします。

  エレメントエディタが開きます。
- “Name”フィールドに `Ki` という名前を入力します。
- **OK** をクリックします。
- 描画エリア内をクリックして、パラメータを配置します。
- “Elements”リストでこのパラメータを右クリックし、ショートカットメニューから **Edit Data** を選択します。

  データコンフィギュレーションウィンドウ（数値エディタ）が開きます。



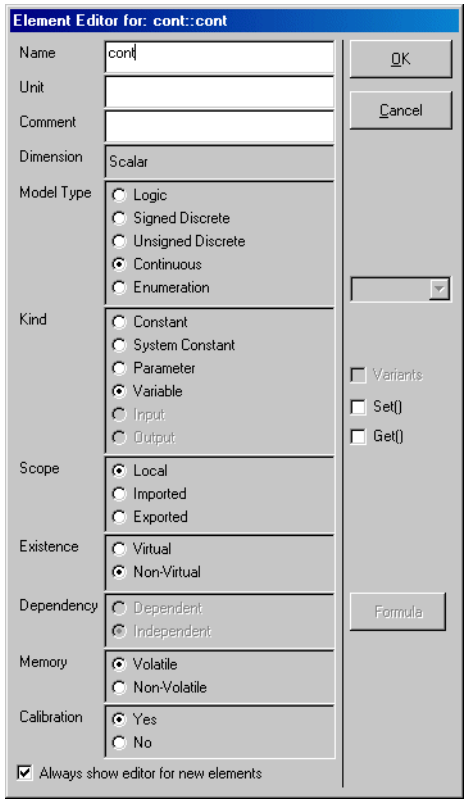
- 入力フィールドに`4.0`と入力してから **<Enter>** を押します。

  この値がパラメータのデフォルト値になります。ダイアグラム内のすべてのパラメータや変数について、このようにしてデフォルト値を割り当てることができます。

依存パラメータを作成する：

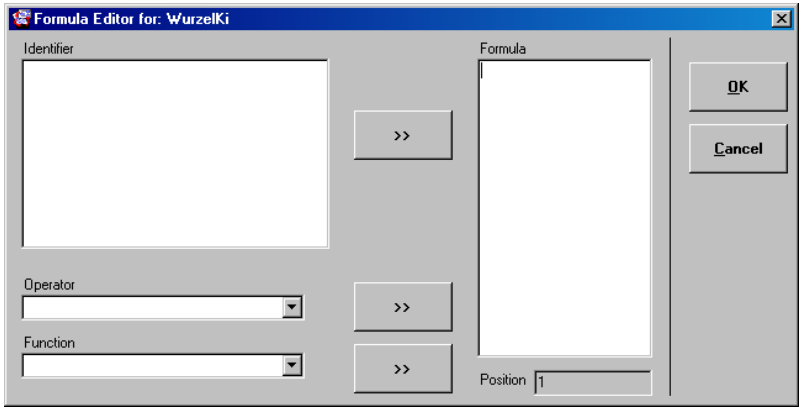
- **Continuous Parameter** ボタンをクリックします。

  エレメントエディタが開きます。



- “Name”フィールドに `WurzelKi` という名前を入力します。
- “Dependency”フィールドの **Dependent** オプションを選択します。

- **Formula** ボタンでフォーミュラエディタを開きます。



- “Identifier”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、**Add** を選択します。

  仮パラメータが作成されます。
- “Identifier”フィールドに新しいパラメータの名前 x を入力します。
- “Formula”フィールドに変換規則を定義します。

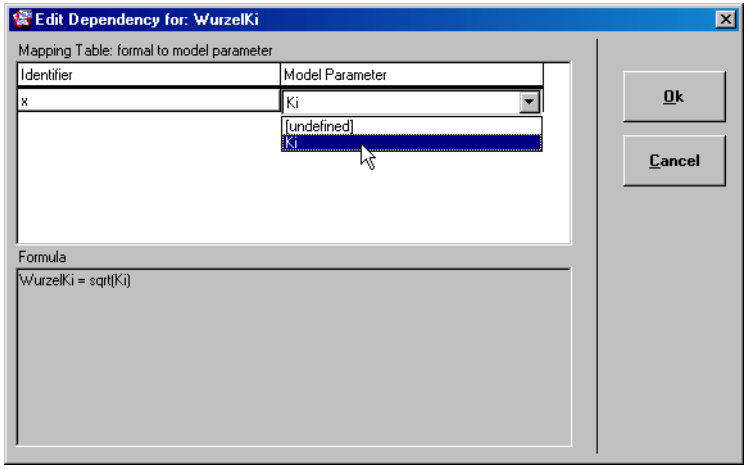
  “Operator”コンボボックスから演算子を選択して、複雑な式を定義することができます。演算子は、“Operator”フィールドのとなりの >> ボタンを押すたびに 1 つずつ“Formula”フィールドに挿入されます。

  同様に、“Function”コンボボックスで関数を選択し、“Function”コンボボックスの隣の >> ボタンを押して“Formula”フィールドに挿入することもできます。
- ここでは例として、仮パラメータの平方根を求める計算を定義します。

  Identifier：x
  Formula：`sqrt(x)`
- **OK** でエレメントエディタを閉じます。

  カーソルが十字カーソルに変わります。
- 描画エリア内をクリックして、パラメータを配置します。

- ブロックダイアグラムエディタで、“Elements” リストの `WurzelKi` を右クリックしてショートカットメニューを開き、**Edit Data** を選択します。
- “Dependency Editor” ウィンドウで、コンボボックス内のモデルパラメータ（この例では `Ki`）を仮パラメータに割り当てます。



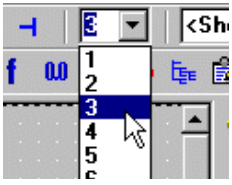
- **OK** をクリックして、データ入力を完了します。

  これで、パラメータ `Ki` に依存し、適合時に `Ki` に基づいて自動的に算出される依存パラメータが定義されました。後で実験を行う時に、この依存関係を確認することができます。

すべてのエレメントを作成したので、次に積分器を定義します。ダイアグラムの残りの部分を以下のようにして完成させてください。

ダイアグラムを作成する：

- “Argument Size” コンボボックスの値を 3 にし、乗算演算子の入力の数を指定します。

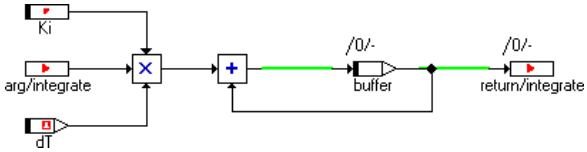


- 乗算演算子を作成し、描画エリアに配置します。

- **dT** ボタンをクリックして dT エレメントを作成します。
- dT エレメントを描画エリアに配置します。
- 2 つの入力を持つ加算演算子を作成し、描画エリアに配置します。

  この演算子を作成する前に、“Argument Size”の値を 2 に戻してください。
- エレメントを下図のように接続します。

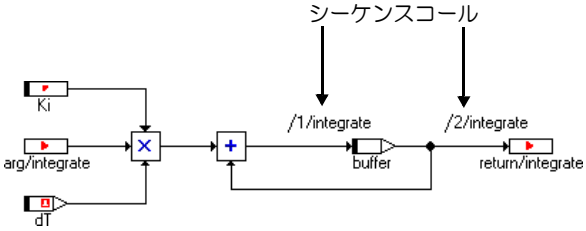
  バッファと戻り値の入力を示す線は、緑色で表示されます。



ここまでで、ダイアグラムのすべてのエレメントの設定が終わりました。次に、シーケンスコールを定定義して、計算の順序を決定します。

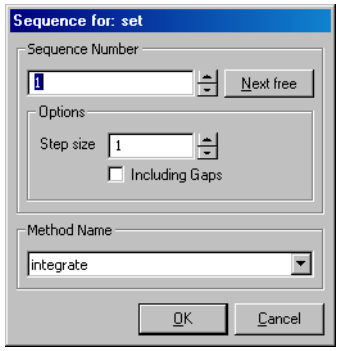
シーケンスコールに値を割り当てる：

- 変数 `buffer` の上に表示されているシーケンスコールを右クリックします。



- ショートカットメニューから **Edit** を選択します。

  シーケンスエディタ（“Sequence”ダイアログボックス）が開きます。



- **OK** をクリックして、デフォルト設定を確定します。

  この設定により、この代入処理は、積分器のアルゴリズムの冒頭部分で行われます。

シーケンスコール内のシーケンス番号を調整する：

- `integrate` の戻り値の上に表示されているシーケンスコールを右クリックします。
- ショートカットメニューから **Edit** を選択します。
- シーケンスエディタで、“Sequence Number”の値を 2 にします。
- **OK** をクリックします。

  この設定により、変数 `buffer` が更新された後に戻り値への代入が行われます。

レイアウトを調整する：

- **Component** → **Edit Layout** を選択します。

  レイアウトエディタが開きます。
- または、“Information/Browse”ビューから“Layout”タブ内のレイアウトをダブルクリックして、レイアウトエディタを開くこともできます。
- 引数を `integrate` からブロックの左側の中程にドラッグします。
- 戻り値をブロックの右側の中程にドラッグします。

- **OK** をクリックします。

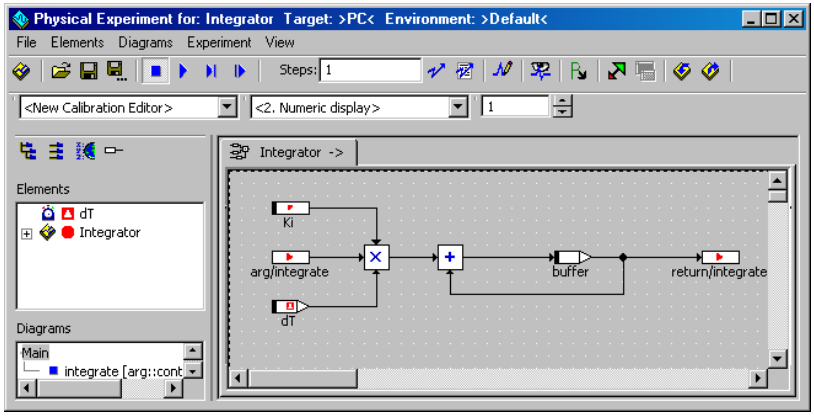
これで、積分器クラスのダイアグラムは完成です。今度は **Diagram → Store To Cache** を選択して、ダイアグラムの変更内容を保存します。ダイアグラム自体に影響を与えない部分の変更内容は、自動的に保存されます。次に、コンポーネントマネージャウィンドウで **File → Save Database** を選択して、変更内容をデータベースに保存します。

6.3.2 積分器の実験を行う

ここでも、初めにイベントジェネレータ、次にデータジェネレータ、最後に測定システムをセットアップします。

積分器用の実験をセットアップする：

- **Component → Open Experiment** を選択して、実験環境を起動します。



- **Event Generator** ボタンをクリックします。
- イベント `integrate` をアクティブにして、`dT` はデフォルト値 0.01 のままにします。
- “Event generator” ウィンドウを閉じます。

- **Data Generator** ボタンをクリックします。
- **Channels → Create** を選択して `integrate` から引数を選択し、`integrate` メソッド用のデータチャンネルを作成します。
- 以下の値を設定します。

Mode： pulse

Frequency： 0.2Hz

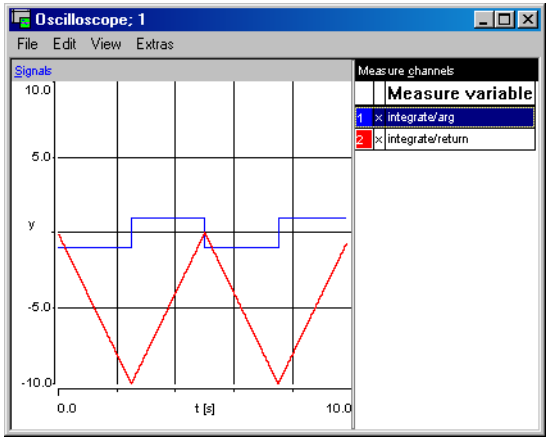
Phase： 0.0s

Offset： -1.0

Amplitude： 2.0

- データジェネレータを閉じます。
- `integrate`メソッドの`arg`と`return`をオシロスコープウィンドウに割り当てます。
- 値軸の範囲を-10～10にし、時間軸の範囲を10秒にします。
- **Start Offline Experiment** を選択して、実験を開始します。

`integrate` メソッドの出力値は、引数が正なら増加し、負なら減少します。入力カーブの正と負の部分の時間は等しいので、安定範囲内の値が出力され続けます。



実験をリセットする：

実験を停止してから再開する際、すべての変数の値は停止された時に保存された値で続行されますが、場合によっては変数を初期値にリセットする必要がある場合もあります。

- **Elements → Reinitialize → Variables**（または **Parameters** 、**Both**）を選択します。

  選択したコマンドに応じて、すべての変数またはすべてのパラメータ、またはその両方が初期値にリセットされます。

次に、実験の条件を変更して、積分器ファンクションのカーブを表示してみましょう。`Ki` パラメータを調整、つまり「適合」し、入力値を変更します。

積分器の実験を行う：

- “Elements”ペーンの `Integrator` の隣の“+”シンボルをクリックして、`Integrator` エレメントを開きます。
- パラメータ `Ki` を選択します。
- **Elements → Calibrate** を選択します。

  このパラメータ用の数値エディタが開きます。
- 5 と入力してから **<Enter>** を押して、このパラメータの値を 5 にします。

  オシロスコープの出力カーブの勾配が急になります。
- 値を 3 にします。

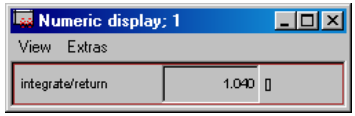
  再び、出力カーブの傾斜が緩くなります。
- パラメータの値を 4 に戻してから数値エディタを閉じます。
- “Data Generator”ウィンドウを開きます。
- 入力パルスのオフセットを -0.5 にします。
- **OK** をクリックします。

これで正の部分の方が大きくなるので、出力値が増加し、ある時点でオシロスコープの表示範囲を超えてしまいます。このような場合、個々の値について、オシロスコープのスケールを変更することができます。また、以下のようにして数値表示ウィンドウを開いて出力値を表示することもできます。

値を数値で表示する：

- 実験環境の“Measure View”コンボボックスから `<1. Numeric Display>` を選択します。
- “Elements”ペーンで、`integrate` メソッドの戻り値 (`return`) を選択します。
- **Elements → Measure** を選択します。

  “Numeric display”ウィンドウに、現在の戻り値が表示されます。



- 依存パラメータ `WurzelKi` も表示します。

- `Ki` の値を変え、**Update Dependent Paramters** ボタンで `WurzelKi` を更新し、実験を行います。

6.3.3 まとめ

このレッスンでは、ASCET で以下の作業を行いました。

- パラメータを作成する
- 依存パラメータを作成し定義する
- 変数を作成する
- 複数の入力を伴う演算子を作成する
- シーケンスコールのシーケンス番号を設定する
- デフォルト値を割り当てる
- 実験中に値を適合する
- “Numeric display”ウィンドウに値を表示する

6.4 実際的な例

このレッスンでは、標準の PI フィルタを若干拡張したものをベースにして、コントローラを作成します。このコントローラは、自動車のアイドリング時のエンジン回転速度を一定に保つために用いられます。

エンジンのアイドリング速度をコントロールする場合、実際の回転数 n がアイドリング時の目標値 `n_nominal` に近い状態を確実に維持する必要があります。`n_nominal` から値 n を引いて、コントロールする偏差を決定します。

この実際の回転数の偏差が、`air_nominal` の値を算出するための基礎になります。`air_nominal` はスロットル位置、つまりエンジンの吸気量を決定します。

6.4.1 コントローラを定義する

コントローラのダイアグラムを作成する手順は、これまでのレッスンで用いた手順と同じです。

- コンポーネントマネージャで新しいフォルダを追加し、コンポーネントを作成します。
- インターフェースを定義し、ブロックダイアグラムを作成します。

これまでのレッスンと大きく異なるのは、コントローラをモジュールとして実装する点です。モジュールはプロジェクトの最上レベルのコンポーネントとして用いられます。モジュール内では、プロジェクトを構成する「プロセス」が定義されます。

コントローラ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、新しいサブフォルダを `Tutorial` フォルダに追加し、その名前を `Lesson4` にします。
- `Lesson4` フォルダを選択し、**Insert → Module → Block diagram** を選択して新しいモジュールを追加します。

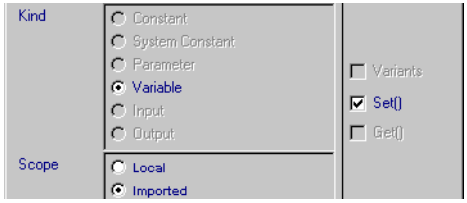
- 新しいモジュールの名前を IdleCon にし、ブロックダイアグラムエディタを開きます。
- “Diagrams” ペーンで、ダイアグラム process の名前を p_idle にします。

モジュールの機能は「プロセス」という単位で定義されます。モジュールにおけるプロセスは、クラスにおけるメソッドに相当します。プロセスはメソッドとは異なり、引数や戻り値を伴いません。ASCET におけるプロセス間の通信、つまりデータ交換は、「受信メッセージ」（入力）および「送信メッセージ」（出力）と呼ばれる方向性のあるメッセージを使用して行われます。

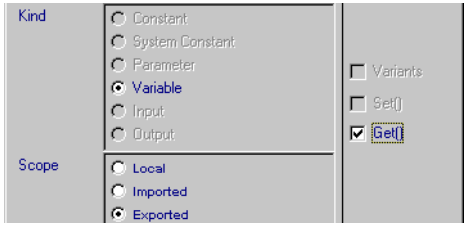
ここで作成するコントローラは、実際の回転数を受信メッセージ n として受け取り、それを使用して算出されたスロットル位置を air_nominal という送信メッセージに代入して出力します。

コントローラのインターフェースを定義する：

- **Receive Message**ボタンをクリックして受信メッセージを作成し、その名前を n にします。
- メッセージ n のエレメントエディタを開き、**Set()** オプションをオンにします。



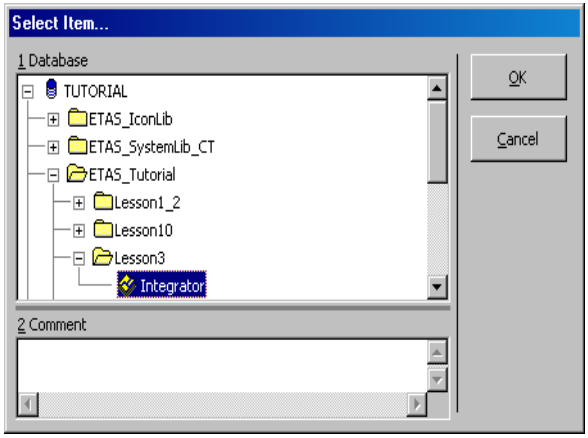
- **Send Message** ボタンをクリックしてから描画エリア内をクリックして、送信メッセージを作成します。
- 送信メッセージの名前を air_nominal にします。
- メッセージ air_nominal のエレメントエディタを開き、**Get()** オプションをオンにします。



このコントローラには、レッスン 3 で作成した積分器 Integrator を使用します。

コントローラに Integrator を追加する：

- **Element → Add Item** を選択して、“Select item” ダイアログボックスを開きます。



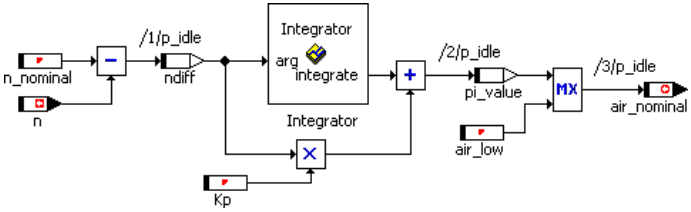
- “1 Database” ペーンの `Tutorial¥Lesson3` フォルダから `Integrator` を選択し、**OK** をクリックして積分器を追加します。
- 積分器がコンポーネント `IdleCon` に内包されます。コンポーネントは参照により内包されるため、この積分器の元の定義を変更すると、他のコンポーネントに内包されるこの積分器のコンポーネントにも、その変更が反映されます。

これまでに追加したエレメント以外に、以下のエレメントをコントローラに追加する必要があります。

- `ndiff` と `pi_value` という、continuous 型の 2 つの変数
- `n_nominal`、`Kp`、および `air_low` という、continuous 型の 3 つのパラメータ

コントローラの残りの部分を定義する：

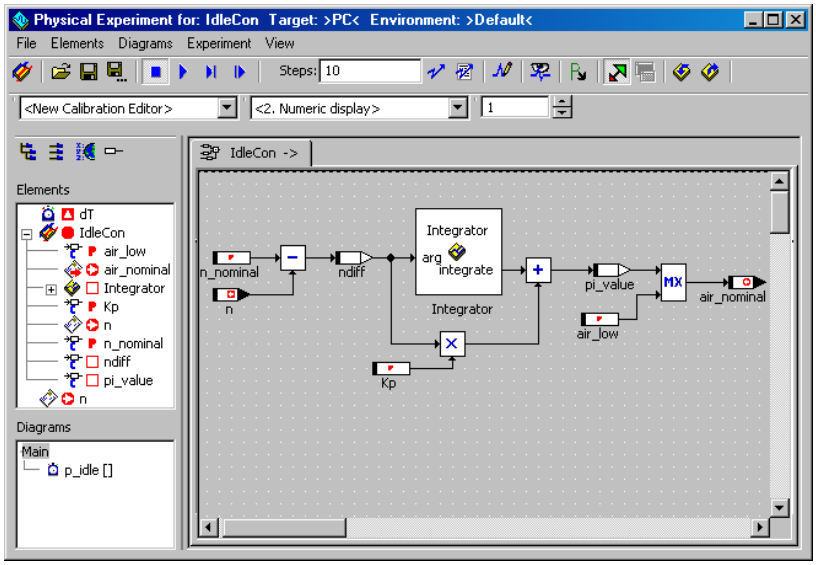
- 必要な演算子や他のエレメントを作成してから、それらを下のブロックダイアグラムのように接続します。



- “Elements”リストから、`n_nominal` パラメータを選択して、**Element → Edit Data** を選択します。
- `n_nominal` の値を 900 にします。
- `Kp` の値を 0.5 にします。
- ダイアグラム内の定義を保存し、変更をデータベースに適用します。

6.4.2 コントローラの実験を行う

モジュールの実験は、コンポーネントの実験と同様の手順で行います。まず、データジェネレータとイベントジェネレータをセットアップし、次に測定システムをセットアップします。



実験をセットアップする：

- **Component → Open Experiment** を選択して、実験環境を起動します。
- “Event Generator”ウィンドウを開き、プロセス `p_idle` 用のイベントをイネーブルにして、`dT` はデフォルト値の 0.01 のままにしておきます。

  プロセス用のイベントも、メソッド用のイベントと同じように機能します。
- “Data Generator”ウィンドウを開き、受信メッセージ n 用のチャンネルに以下の値を設定します。

Mode：	pulse
Frequency：	1.0Hz
Phase：	0.0
Offset：	800.0
Amplitude：	200.0

- 変数 n_diff および air_nominal をオシロスコープに割り当てます。
- オシロスコープの値軸の範囲を -500 ～ 500 に、時間軸の範囲を 2 にします。
- **Save Environment** ボタンをクリックします。

これで実験のセットアップが終わり、回転数の偏差とスロットル位置の関係を表示できるようになりました。

コントローラの実験を行う：

- **Start Offline Experiment** ボタンをクリックして、実験を開始します。
- 変数Kiおよび Kp 用の適合ウィンドウを開きます。そこから、値 Ki および Kp を調整し、その出力に対する影響を調べることができます。

  有意義な値に戻すために、実験中にモデルを再初期化する必要があるかもしれません。

6.4.3 プロジェクト

プロジェクトは、完結したソフトウェアシステムとして機能する ASCET ソフトウェアの単位です。プロジェクトは、実験ターゲットやマイクロコントローラターゲットを使用して、オンラインでリアルタイムな実験を行うことができます。個々のコンポーネントの実験はオフライン（PC ベース）でしか行えません。

実験はプロジェクト単位で実行されます。プロジェクト用のコードが生成されると、オペレーティングシステムコードも必ず生成されます。ASCET で作成されたソフトウェアシステムをリアルタイムに実行するには、オペレーティングシステムのセットアップが必要です。ソフトウェアシステムをリアルタイムに実行する実験を「オンライン実験」と呼びます。これまでのレッスンで行った実験はすべてオフライン実験であり、リアルタイムなシミュレーションは行われませんでした。

注記

オンライン／オフラインにかかわらず、ASCET の実験は実際にはすべてプロジェクトの単位で行われます。このことはオフライン実験でデフォルトプロジェクトを使用する（このプロジェクトはユーザーからは見えない場合もあります）ことでも明らかです。オペレーティングシステムを定義する目ために明示的にプロジェクトを作成してセットアップしなければならないのは、オンライン実験の場合だけです。ただし、ユーザー独自のアプリケーション用にデフォルトプロジェクトを設定することもできます。

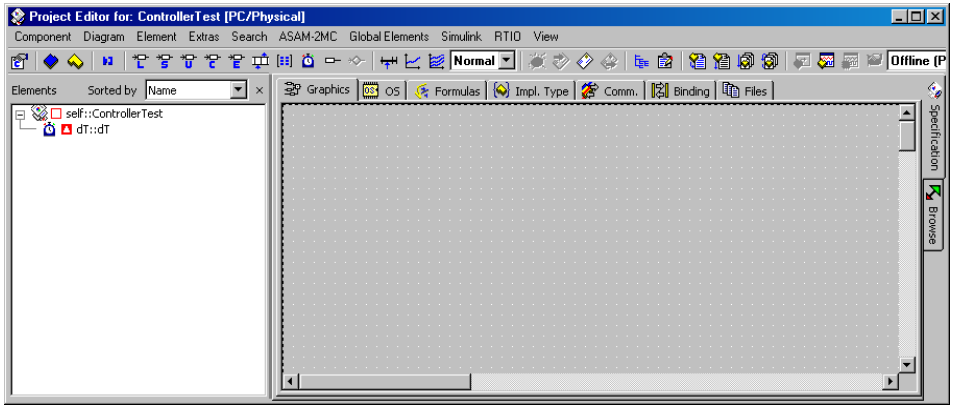
6.4.4 プロジェクトをセットアップする

コンポーネントマネージャで、IdleCon モジュールと同じフォルダにプロジェクトを作成します。

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、新しいプロジェクトを追加します。
- プロジェクトの名前を ControllerTest にします。
- **Component → Edit** を選択するか、またはプロジェクトエレメントをダブルクリックします。

  プロジェクト用のプロジェクトエディタが開きます。



次に、プロジェクトの“Elements”リストに IdleCon コントローラを追加します。

プロジェクトにコンポーネントを追加する：

- プロジェクトエディタで **Element → Add Item** を選択して、“Select Item”ダイアログボックスを開きます。
- “1 Database”リストから、Tutorial/Lesson4 フォルダのコンポーネント、IdleCon を選択します。

- **OK** をクリックして、このコンポーネントを追加します。

  コンポーネントの名前が、プロジェクトエディタの“Elements”リストに表示されます。

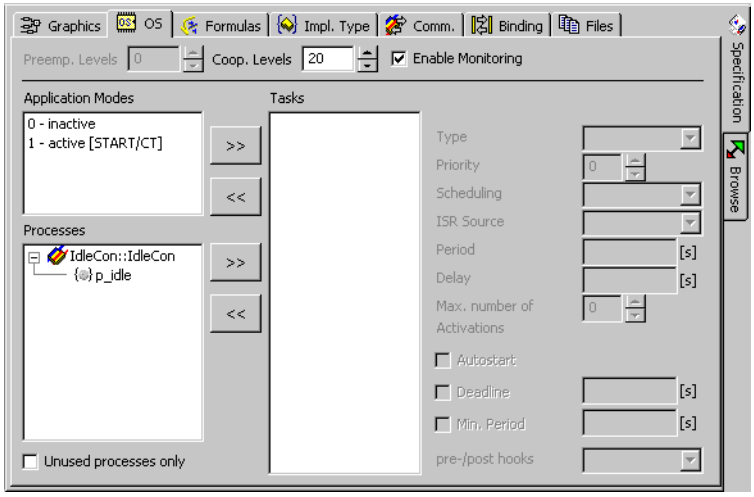
参照されるコンポーネントは、非参照コンポーネントに内包されます。つまり、内包されたコンポーネントのダイアグラムを変更すると、その変更がプロジェクト全体に反映されます。

プロジェクトに含まれるタスクとプロセスのスケジューリングは、オペレーティングシステムによって行われます。プロジェクト用のコードを生成する前に、タスクをいくつか作成してそれらにプロセスを割り当てておく必要があります。

オペレーティングシステムのスケジューリング条件は、プロジェクトエディタの“OS”タブに定義します。ここで、`p_idle` プロセスが 10ms ごとに起動されるように、オペレーティングシステムのスケジュールを定義しましょう。

プロジェクト用のオペレーティングシステムのスケジュールをセットアップする：

- “OS”タブをクリックします。



- **Task → Add** を選択して、新しいタスクを作成します。
- 作成したタスクの名前を `Task10ms` にします。

  作成されるタスクは、デフォルトでは Alarm タスク、つまり、オペレーティングシステムにより周期的に起動されるタスクです。

- “Period”フィールドで、このタスクの周期を 0.01 秒にします。

  この周期は、タスクが起動される頻度を規定します。この例では、10ms ごとに起動されます。

- “Processes”リストの `IdleControl` というアイテムの隣の“＋”シンボルをクリックして、このアイテムを開きます。

- プロセス `p_idle` を選択し、**Process → Assign** を選択します。

  このプロセスが `Task10ms` タスクに割り当てられ、“Tasks”リストのこのタスク名の下に表示されます。

プロジェクト内においては、インポートエレメントやエクスポートエレメントを用いてプロセス間通信を行います。これらのエレメントはグローバルエレメントで、モジュール間通信の送信メッセージと受信メッセージに相当します。グローバルエレメントはプロジェクト単位で宣言され、プロジェクト内の各モジュールに含まれる同名のエレメントに結び付けられていなくてはなりません。

グローバルエレメントを定義する：

- プロジェクトエディタで、**Global Elements → Resolve Globals** を選択します。

  グローバルエレメントが作成され、それぞれ対応するエレメントに結び付けられます。同じ名前のエレメント同士が自動的に結び付けられます。

  送信メッセージは、デフォルトでエクスポートエレメントとしてモジュール内に定義されます。

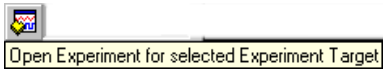
6.4.5 プロジェクトの実験を行う

まず、このプロジェクトのオフライン実験を行います。オフライン実験は、ハードウェアを接続せずに PC 上で行うことができます。プロジェクトのデフォルト設定では PC 上で稼働するようになっているので、この設定を変更する必要はありません。プロジェクトのオフライン実験は、コンポーネントのオフライン実験と同様の手順で行います。

実験をセットアップする：

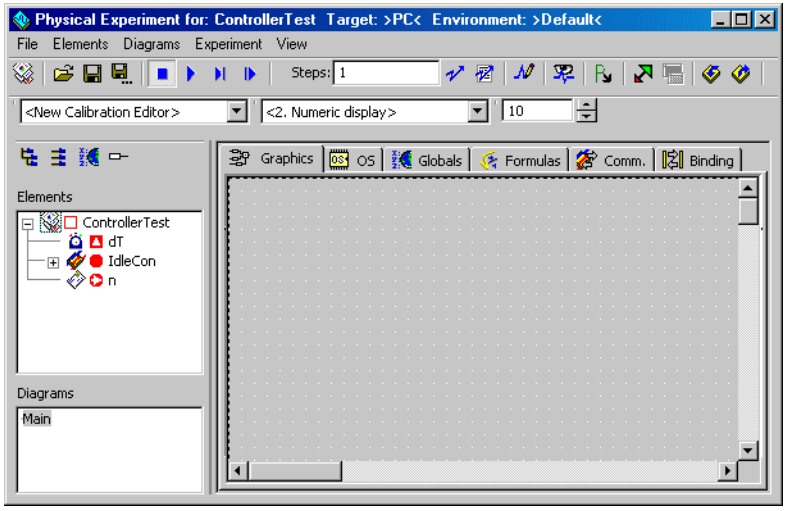
- コンポーネントマネージャで、**File → Save Database** を選択します。

  実験環境を起動する前に、変更内容を必ずデータベースに保存することをお勧めします。



- **Open Experiment for selected Experiment Target** ボタンをクリックします。

プロジェクトのコードが生成され、オフライン実験が開きます。



- **Open Event Generator** ボタンをクリックします。

  プロジェクト用のイベントジェネレータには、コンポーネントの実験の場合のように各メソッドや各プロセス用のイベントではなく、実験で使用する各タスク用のイベントが表示されます。
- ダイアログボックスからタスク `generateData` をイネーブルにし、`dT` 値にはデフォルトの 0.01 秒を採用します。

  タスク `Task10ms` はデフォルト状態ですでにイネーブルになっているので、これでタスク generateData および 10ms のイベントの `dT` 値はともに 0.01 秒になります。これ以上変更の必要はありません。
- イベントジェネレータを閉じます。
- データジェネレータと測定システムを、前回の実験と同じ値でセットアップします（139 ページの「コントローラの実験を行う」を参照してください）。
- **File → Save Environment** を選択して、環境設定を保存します。

実験を行う：

- **Start Offline Experiment** ボタンをクリックします。
- 前項と同様にKiおよびKpパラメータを調整して、出力への影響を確認します。

6.4.6 まとめ

このレッスンでは、ASCET で以下の作業を行いました。

- モジュールを作成する
- モジュール内のメッセージを作成する
- コンポーネントマネージャで作成したコンポーネントをブロックダイアグラムに組み込む
- プロジェクトを作成する
- プロジェクトにコンポーネントを内包する
- タスクを作成し、それらにプロセスを割り当てる
- プロジェクトの実験を行う

6.5 プロジェクトを拡張する

このレッスンでは、コントローラを少し改良して一層実用的なものにします。センサで読みとった値を実際の値に変換する、シグナルコンバータを作成します。たとえば自動車制御アプリケーションで用いられるような多くのセンサは、温度、位置、毎分の回転数などの測定値に対応する電圧を返します。この電圧と測定値の関係は、必ずしも一次関数で表せるとは限らないので、ASCET では、この種の対応関係を効率的にモデリングできる特性テーブルを使用できます。

6.5.1 シグナルコンバータを定義する

シグナルコンバータをモデリングするために、まずフォルダとモジュールを作成して、機能を定義します。シグナルコンバータは 2 つの特性カーブを使用して、入力値に対する出力値を求めます。

モジュールを作成する：

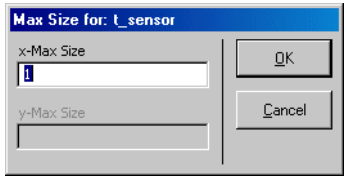
- コンポーネントマネージャで、新しいフォルダ `Tutorial¥Lesson5` を作成します。
- 新しいモジュールを作成し、その名前を `SignalConv` にします。
- **Component → Edit** を選択するか、またはエレメントをダブルクリックして、ブロックダイアグラムエディタを開きます。

- ブロックダイアグラムエディタで **Diagram → Add Process** を選択して 2 つめのプロセスを作成します。
- 2 つのプロセスの名前を `n_sampling` および `t_sampling` にします。
- “Elements”リストに、2 つの受信メッセージ `U_n` および `U_t` と、2 つの送信メッセージ `t` および `n` を作成します。

- **One-D Table Parameter** ボタンをクリックして、特性カーブエレメントを作成します。

  エレメントエディタが開きます。
- `t_sensor` という名前を入力し、エレメントエディタを閉じます。

  続いて“Max size”ダイアログボックスが開きます。特性カーブは 1 次元なので、“y-Max Size”フィールドは無効になっています。

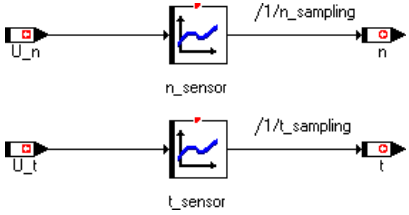


- “x-Max Size”フィールドに値 13 を入力します。

  これにより、この特性カーブを最大 13 列まで広げることができるようになりました。
- **OK** をクリックして、ダイアログボックスを閉じます。
- 描画エリア内をクリックして、テーブルを配置します。

  このテーブルが“Elements”リストに追加されます。
- 最大 2 列の第 2 のテーブルを作成し、`n_sensor` という名前にします。

- 下図のようにエレメントを接続し、シーケンシングを編集して、この処理を行うプロセスを割り当てます。



次に、2 つの特性カーブのデータを編集します。ASCET ではテーブルエディタを使用して各種テーブルデータ（配列、特性カーブ、および特性マップ）を編集します。

テーブルを編集する：

- テーブル t_sensor を右クリックし、ショートカットメニューから **Edit Data** を選択します。

  テーブルエディタが開きます。
- テーブルのサイズを以下のように調整します。

  テーブルが 13 列に拡張され、z 値はすべてデフォルトで 0 になります。
- 以下の値を入力します。上の行が X 行、下の行が Z 行の値です。

0.00	0.08	0.30	0.67	1.17	2.50	5.00	7.50	8.83	9.33	9.70	9.92	10.00
-40.0	-26.0	-13.0	0.0	13.0	40.0	80.0	120.0	146.0	160.0	173.0	186.0	200.0

  まずサンプルポイント（X 値）を左から右の順に入力して、テーブルを編集します。
- 編集しようとする X 値をクリックしてから、ダイアログボックスに新しい値を入力します。

  新しい値は両隣のサンプルポイントの間の値でなければなりません。
- 次に、Z 値（出力値）をクリックし、強調表示された値の上に適切な値を入力します。

- 同じ方法で、以下のデータを使用して、第 2 のテーブルを編集します。

  0.0　　10.0

  0.0　　6000.0

- ブロックダイアグラムエディタで、**Diagram → Store to Cache** を選択します。
- コンポーネントマネージャで、**Save** ボタンをクリックして、変更内容を保存します。

この例では、第 2 のテーブルは出力が入力の変化に従って直線的に変化する関係を表せばよいので、必要なサンプルポイントは 2 つだけです。値の補間モードとして線形補間を指定したので、これで充分に機能します。

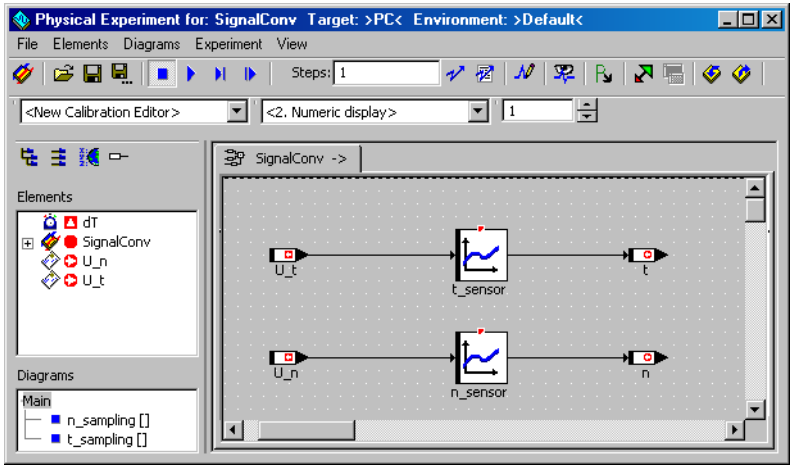
線形補間では、2 つのサンプルポイント間の入力値に対応する出力値は、直線から求められます。この場合、入力値が 0 なら 0 が返され、10 なら 6000 が返されます。入力値が 5 の場合、戻り値は補間により 3000 になります。

6.5.2 シグナルコンバータの実験を行う

新しいコンポーネントの実験を行い、テーブルによる変換処理の結果を調べてみましょう。2 つのテーブルの値の範囲は互いに異なるので、それぞれに専用のオシロスコープウィンドウを使用します。

実験をセットアップする：

- コンポーネントマネージャから **Component → Open Experiment** を選択して、実験環境を開きます。

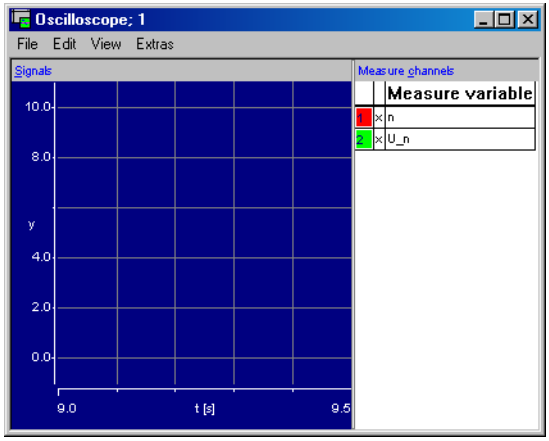


- コンポーネント内の各プロセス（n_sampling、t_sampling、generateData）用にイベントを作成し、各イベントの dT 値を 4ms にします。
- データジェネレータで、メッセージ U_n 用と U_t 用にチャンネルを 1 つずつ作成し、両チャンネルに以下の値を設定します。

Mode：	sinus
Frequency：	2.0Hz
Phase：	0.0
Offset：	5.0
Amplitude：	5.0

- メッセージ n および U_n のオシロスコープウィンドウと、メッセージ t および U_t のオシロスコープウィンドウを作成します。

  後者のオシロスコープを作成する際には、“Select Measure View” コンボボックスで必ず <2. Oscilloscope> を選択してください。



2 つのテーブルのサンプリングポイントの分解能とそれに対応する補間値は大きく異なるので、各チャンネルの表示設定は、それぞれのオシロスコープ内で個別に行う必要があります。

オシロスコープを測定値にあわせて調整する：

- プロセス n_sampling 用のオシロスコープ（チャンネル U_n および n）を選択します。

- “Measure Channels” リストから、メッセージ n を選択し、**Extras → Setup** を選択します。

  メッセージ n 用の “Display Setup” ダイアログボックスが開きます。
- 値軸の範囲を 0 ～ 6000 にし、時間軸の範囲を 0.5 にします。
- メッセージ U_n 用の “Display Setup” ダイアログボックスを開きます。
- その値軸の範囲を -1 ～ 11 にします。

  時間軸の範囲は、1 つのオシロスコープウィンドウ内のすべての変数について同じでなければならないので、時間軸の範囲を変更する必要はありません。
- プロセス t_sampling（チャンネル U_t および t）用のオシロスコープを選択し、そのチャンネルを以下のようにセットアップします。

	U_t	t
Min	-1	-40
Max	11	200
Extent	0.5	0.5

- **Save Environment** ボタンをクリックして、環境設定を保存します。

これで、実験を実行してシグナルコンバータの機能を調べることができるようになりました。2 つの変換処理の相違を調べてみましょう。

実験を行う：

- **Start Offline Experiment** ボタンをクリックします。

  n_sensor テーブルでは、入力される正弦波の振幅だけが変化します。ここでの入力は 0 ～ 10 ボルトの電圧信号です。これが 0 ～ 6000rpm の回転速度にマッピングされます。

  テーブル t_sensor では、入力される電圧と出力される温度との関係は一次関数では表せません。この関係は自動車制御用に一般に用いられている温度センサの応答特性カーブと同じです。
- データジェネレータからの入力をさまざまな波形に変更して、両方の出力カーブに表れる影響を調べます。

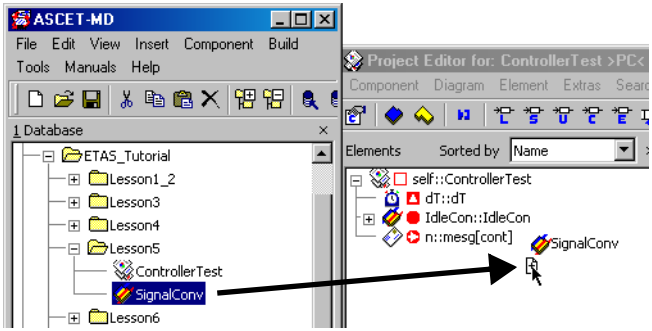
6.5.3 シグナルコンバータをプロジェクトに統合する

シグナルコンバータの定義が終わったので、これを、レッスン 4 で作成したプロジェクトに統合します。シグナルコンバータからの出力信号が、エンジンコントローラへの入力信号として使用されます。

プロジェクトにシグナルコンバータを統合するために、新しいプロセス用のタスクをセットアップし、プロセス間通信に必要なグローバルエレメントを宣言してその結び付けを行います。

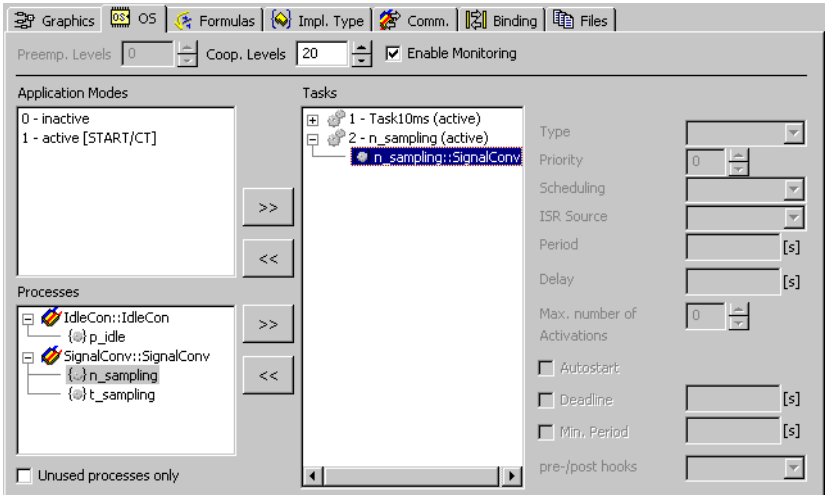
シグナルコンバータをプロジェクトに追加する：

- コンポーネントマネージャから、プロジェクト `ControllerTest` 用のプロジェクトエディタを開きます。
- モジュール `SignalConv` を、コンポーネントマネージャの“1 Database”リストからプロジェクトの“Elements”リストにドラッグします。



- “OS”タブをクリックして、オペレーティングシステムエディタを起動します。
- 新しいタスク `n_sampling` を作成します。
- 新しいタスクの周期を 0.004 秒にします。

- プロセス n_sampling をタスク n_sampling に割り当てます。



これで、プロジェクトに 2 つのタスクが定義されました。第 1 のタスクは 10 ミリ秒ごと、第 2 のタスクは 4 ミリ秒ごとに起動されます。1 つのタスクに割り当てられているすべてのプロセスは、そのタスクに定義されているインターバルで実行されます。この例ではそれぞれのタスクにプロセスが 1 つずつしかありませんが、必要に応じて任意の数のプロセスを割り当てることができます。

シグナルコンバータを統合するためには、次にモジュール間の通信を解決します。プロセス間通信はグローバルエレメントを介して行われます。プロジェクト内で用いられるすべてのグローバルエレメントは、対応するモジュール内でメッセージとして定義されていなければなりません。

デフォルトでは、送信メッセージはモジュール内に定義されますが、受信メッセージは、通常モジュールにインポートされるものであるため、プロジェクト内で受信メッセージを定義する必要があります。

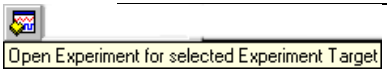
グローバルエレメントをセットアップする：

- **Global Elements → Resolve Globals** を選択して、結び付けを自動で行います。
- **Global Elements → Delete Unused Globals** を選択して、前のレッスンで使用した結び付けを削除します。

必要なグローバルエレメントはすべて、自動的に作成され、同じ名前のエレメントに結び付けられます。たとえばグローバルメッセージ U_n は、SignalConv 内のメッセージ U_n に自動的に結び付けられます。

レッスン 4 ではメッセージ n をプロジェクトのグローバルメッセージとして定義しましたが、このレッスンではメッセージ n をモジュール SignalConv に定義し、モジュールのプロセス間通信に使用します。そのため、使用しないグローバルメッセージ n を削除する必要があります。

プロジェクトの実験を行う：



- **Open Experiment for selected Experiment Target** ボタンをクリックします。
- イベントジェネレータを開き、タスク n_sampling をイネーブルにします。
- このタスクの dT 値を 4 ミリ秒にします。

  プロジェクトのオフライン実験では、オンライン実験時にはオペレーティングシステムが行うスケジューリングを、イベントジェネレータがシミュレートします。
- データジェネレータを開き、既存のデータチャンネルを削除します。
- メッセージ U_n 用に新しいチャンネルを設定します。
- チャンネル U_n を以下のように設定します。

Mode：	pulse
Frequency：	1.0Hz
Phase：	0.0
Offset：	4/3
Amplitude：	1/3

- 回転速度センサの出力電圧 U_n をアクティブにします。

  シグナルコンバータは、特性テーブル n_sensor を使用してこの電圧値を回転数 n に変換します。

  上記の値により、n が前の実験（信号処理なし）と同じ範囲で出力されます。

- **Save Environment** ボタンをクリックします。
- 実験を開始します。

出力されるカーブは、信号処理を行わない例で出力されたカーブと同じはずです。データジェネレータによりスティミュレートされる値は異なりますが、テーブルで処理され、前の実験と同じ出力値になります。

6.5.4 まとめ

このレッスンでは、ASCET で以下の作業を行いました。

- 特性カーブエレメントを作成して使用する
- コンポーネントをプロジェクトに追加する
- プロジェクト内のコンポーネント間の通信を定義する

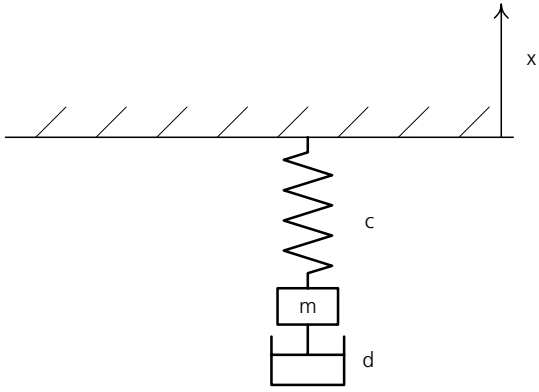
6.6 連続系をモデリングする

物理、機械、電子、およびメカトロニクスに関する処理を現実的にモデリングするには微分方程式を用いることが多く、連続系メソッドが要求されます。このようなメソッドをこれまでの章で作成したプロジェクトに統合する前に、本章では詳しい例を用いて、連続系のモデリングについて説明します。

ASCET は、いわゆる「CT ブロック」による連続系のモデリングとシミュレーションをサポートしています。CT は“Continuous Time”（連続時間）の略で、疑似連続的な時間ステップで計算処理が行われることを意味します。ASCET における連続系モデリングは、連続系の設計に用いられる標準的な記述形式である、状態空間表現をベースとしています。この形式では、CT 基本ブロックを非線形 1 次常微分方程式と非線形出力方程式で定義することができます。ASCET は、これらの微分方程式の最適な解を見つけるための、いくつかのリアルタイム積分メソッドを提供します（ASCET ユーザーズガイド「微分方程式の解法 — 積分のアルゴリズム」の章を参照してください）。

以下に、ばね－質量系の、地球の重力による減衰を伴う動きを例にして、連続系のモデリングの手順を説明します。

6.6.1 運動方程式



上図の質量 m には、以下の力が働いています。

- 重力：$F_g = -mg$
  （g = 重力の加速度）
- ばねの力：$F_F = -c\ (x + l_0)$
  （c = ばね定数、l_0 = 静止時のばねの長さ、x = 質量 m の位置）

- 減衰 F_D = - d x′
  （d = 減衰定数、x′ = 質量の速度）

これにより、以下の運動方程式が得られます。

mx′′ = -mg + F あるいは x′′ = -g + F/m (ただし F = F_F + F_D)

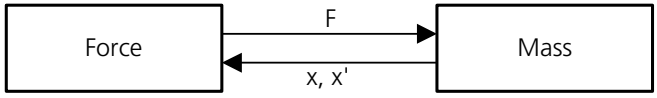
この 2 次微分方程式を（x = x、v = x′ により）2 つの 1 次微分方程式にすると、以下のようになります。

x′ = v
v′ = -g + F/m

以下のモデル設計では、これらの微分方程式を使用します。

6.6.2 モデル設計

ばね－質量系のモデルは、CT ブロックを 1 つだけ用いてシンプルに設計することもできますが、ここでは 2 つの CT ブロックを用いてこのモデルをモデリングします。その過程で「直接通過」や「間接通過」のプロパティについて説明し、これらのプロパティを適切に設定して算術ループを避ける方法を紹介します。



- Force ブロックでは、質量 m の位置と、速度 x′ から求められる摩擦力に基づいて、ばねの力 F を算出します。
- Mass ブロックでは、ばねの力 F から加速度 x′′ を算出します。x′′ を積分して、速度 x′ と位置 x を算出します。

一見して、このシステムでは、どちらのブロックも相手のブロックから要求される出力を算出するために相手のブロックからの入力が必要なので、算術ループになってしまうように見えます。

このループは、「直接通過 」または「間接通過」のプロパティを適切に設定することによって回避することができます。

- Force ブロックでは、下の方程式により算出される出力変数 F は、入力変数 x および x′ に直接依存しています。したがって、このブロックは直接通過として定義されます。

  F = -c(x + l_0) - dx′

- 一方、Mass ブロックでは、出力変数 x および x′ は入力変数 F に直接依存しているわけではなく、ブロックの内部ステート変数に依存しています。これらは、少なくとも初めは初期値になっているので、入力変数 F の値がわ

からなくても、初期値に基づいて出力変数 x および x′ を算出することができます。F の値がわかっている場合には、出力変数は微分方程式を用いて算出されます。

```
x′ = v
v′ = -g + F/m
```

したがって、このブロックは、間接通過として定義します。

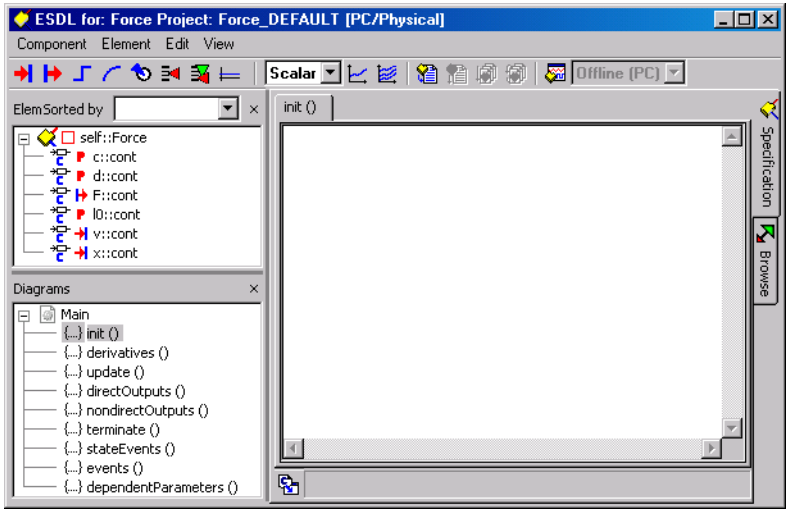
モデルを作成する：

- コンポーネントマネージャで **Insert** → **Folder** を選択してフォルダを作成し、その名前を Lesson6 にします。
- このフォルダ内で **Insert** → **Continuous Time Block** → **ESDL** を選択して、Force および Mass というブロックを作成します。
- Force ブロックをダブルクリックして、ESDL エディタを開きます。

- **Input** ボタンをクリックして、x および v という 2 つの入力（continuous 型）を作成します。

- **Output** ボタンをクリックして、出力 F（continuous 型）を作成します。

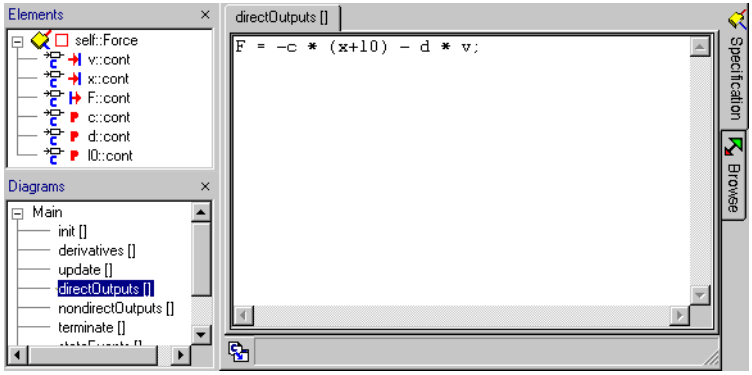
- **Parameter** ボタンをクリックして、定数 c（ばね定数）、d（減衰定数）、l_0（静止時のばねの長さ）を作成します。

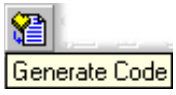


“Diagrams”ペーン内のメソッドはデフォルトで作成されます。

- “Elements”ペーン内の各定数を、クリックして順に強調表示します。
- 強調表示された定数を右クリックして、ショートカットメニューを開きます。
- **Edit Data** メニューアイテムを選択します。

  “Numeric Editor”ダイアログボックスが開きます。
- 定数に現実的な値を割り当てます（例、ばね定数 `c` には 5.0、減衰定数 `d` には 1.0、静止時のばねの長さ l_0 には 2.0）。
- “Diagrams”ペーン内のメソッド `directOutputs[]` をクリックし、“Edit”フィールドに、力を算出する式 `F = -c * (x + l0) - d*v;` を定義します。

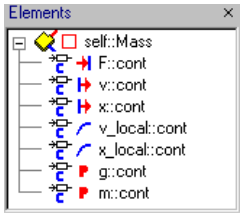




- **Generate Code** ボタンをクリックします。

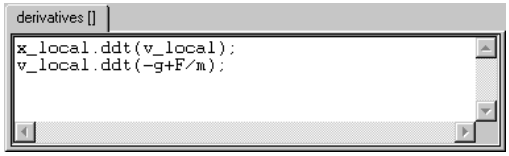
  CT ブロック `Force` がコンパイルされます。
- `Mass` ブロックをダブルクリックして、ESDL エディタを開きます。
- Force ブロックの場合と同様にして、入力 `F`、2 つの出力 `x` および `v`、パラメータ `m`（質量）、および定数 `g`（重力加速度）を作成します。
- Force ブロックの場合と同様にして、`g` および `m` に値を割り当てます（`g` は 9.81、質量 `m` はたとえば 2.0）。

- **Continuous State** ボタンをクリックして、出力を内部で計算するためのステート変数 `x_local` および `v_local` を作成します。

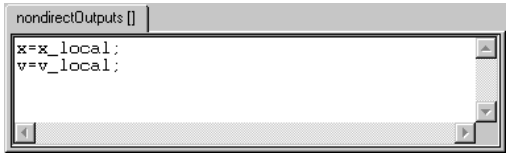


- `derivatives[]` メソッドに、計算に必要な微分方程式を定義します。

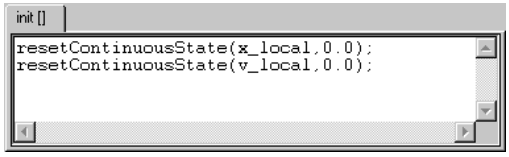
```
x_local.ddt(v_local)
v_local.ddt(-g + F/m)
```

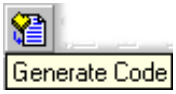


- `nondirectOutputs[]` で、ステート変数 `x_local` および `v_local` を出力 x および v に渡すようにします。



- `init[]` メソッドでは、ユーザーが `resetContinuousState()` 関数を使用して、x と v に現実的な初期値を与えることができます。





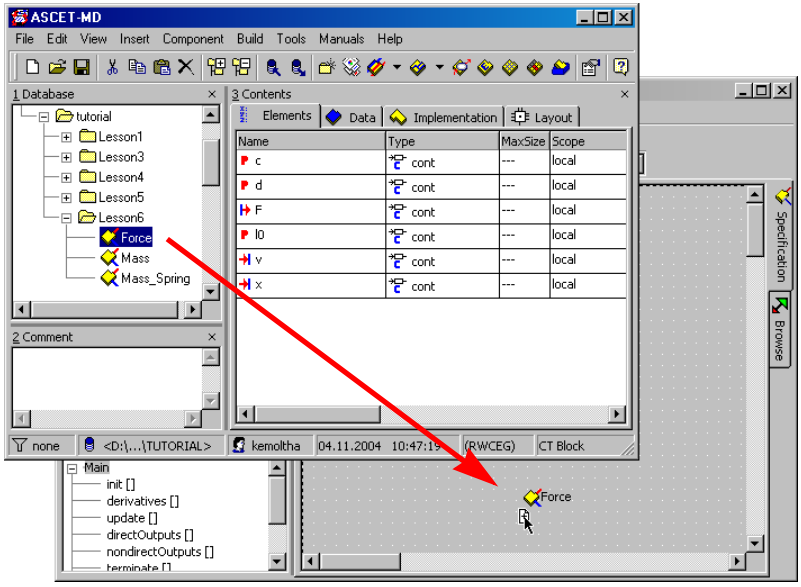
- **Generate Code** ボタンをクリックします。

  CT ブロック `Mass` がコンパイルされます。

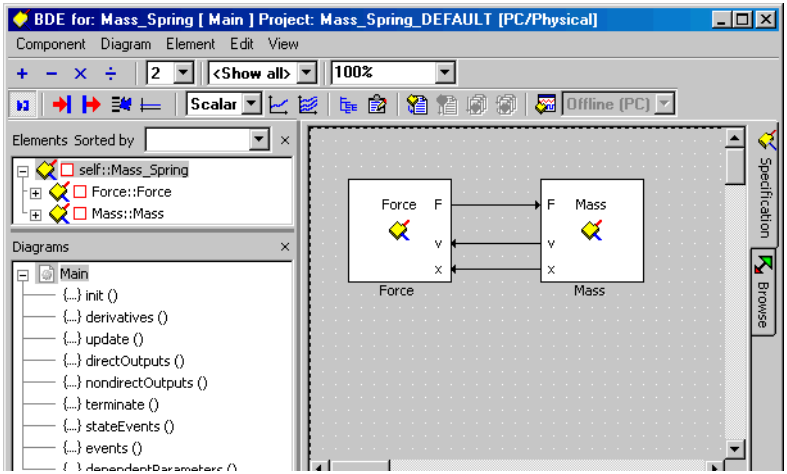
ブロックダイアグラムエディタ（BDE：**B**lock **D**iagram **E**ditor）を使用して、2 つの基本 CT ブロックを結合して 1 つの CT 構造ブロックにします。

2 つの基本 CT ブロックを結合する：

- コンポーネントマネージャで作業ディレクトリに移ります。
- **Insert → Continuous Time Block → Block Diagram** で、新しいブロック `Mass_Spring` を作成します。
- 新しいブロックをダブルクリックしてブロックダイアグラムエディタ（BDE）を開きます。
- コンポーネントマネージャから `Mass` および `Force` ブロックを 1 つずつドラッグして BDE ウィンドウにドロップし、`Mass_Spring` 内に組み込みます。

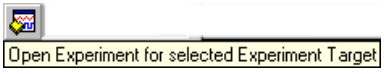


- 対応する入力と出力とを接続します。



注記

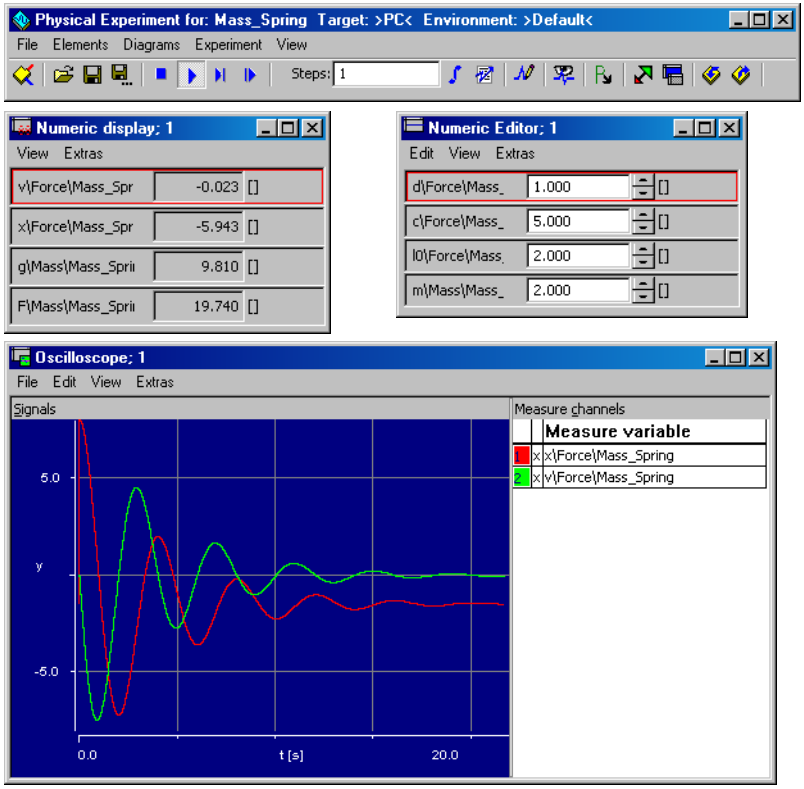
CT 基本ブロックの 1 つをダブルクリックすると、そのブロックを編集できるようになります。ブロックに修正を加えるとライブラリ全体、つまり、その基本ブロックを使用するすべての構造ブロックに影響します。



- **Open Experiment for selected Experiment Target** ボタンをクリックします。

  CT ブロックがコンパイルされ、実験環境が起動されます。

- 以下のように数値エディタとオシロスコープを設定します。



- オシロスコープ内のチャンネルのスケールを、x は -10 ～ 0、v は -8 ～ +8 に調整します。

6.6.3 まとめ

このレッスンでは、ASCET で以下の作業を行いました。

- モデルを作成してプロセスをシミュレートする
- ESDL エディタを使用して、直接および間接通過の CT ブロックを作成する
- ブロックダイアグラムエディタを使用して、複数の CT ブロックを結合する
- 物理実験を行う

6.7 プロセスモデル

前のレッスンでの CT ブロックの紹介に続いて、それらの CT ブロックを使用したコントローラのテストを行います。ASCET では、制御対象となる物理プロセスのモデルを開発し、閉制御ループによってコントローラモデルの実験を行うことができます。これにより、実際の車両を使用する前にコントローラのテストを入念に行っておくことができます。

ここでは、モータの物理プロセスを扱います。この物理プロセスモデルはエンジン回転速度センサの値 `U_n` を返します。そしてコントローラはこの値を処理し、値 `air_nominal` を返します。コントローラの出力値によりエンジンのスロットル位置が決まり、さらにこのスロットル位置が回転速度に影響します。

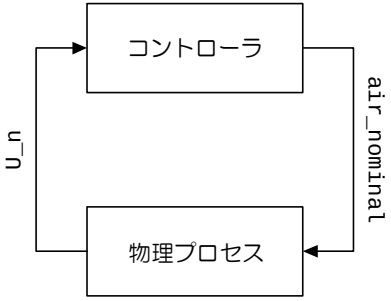


図 6-1 閉ループ実験

このプロセスモデルには CT ブロックを使用します。連続系コンポーネントはプロセスモデルに特に適していて、モデルのベースとなるのは、PT2 系をモデリングする以下の微分方程式です。

$$T^2\ s'' + 2DTs' + s = Ku$$

式 6-1 PT2 系

この方程式においては、パラメータ T、D、および K に適切な値を設定する必要があります。

6.7.1 プロセスモデルを定義する

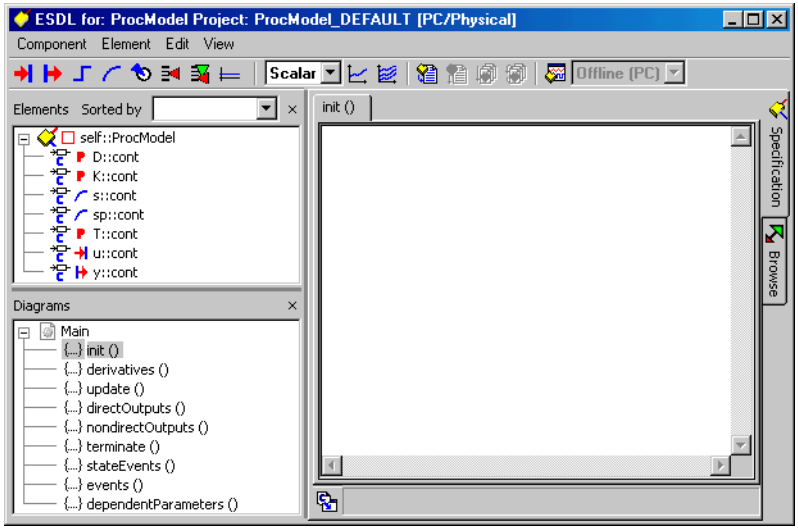
連続系コンポーネントの作成方法は、他のコンポーネントの作成方法とは異なります。連続系コンポーネントには入力と出力があり、これらは引数と戻り値に相当します。主な違いは、連続系ブロックは複数の入力と出力を伴うことができ、それらが特定のメソッドと結びついていないことです。各連続系ブロック内には、あらかじめ一連のメソッドが固定的に定義されていて、これをユーザーが変更することはできません。

ここでは、ESDL コードを使用します。ESDL コードの構文は C++ や Java に似ています。オブジェクトのメソッドは、「オブジェクト名 . メソッド名 (引数); 」の形式で呼び出されます。微分に用いるメソッドは `ddt()` と呼ばれます。たとえば、方程式 $sp = \dot{s}$ は、ESDL では `s.ddt(sp);` という文で表記されます。

連続系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フォルダ `Tutorial¥Lesson7` を作成します。
- 連続系ブロックを追加するために、**Insert → Continuous Time Block → ESDL** を選択します。
- 新しいコンポーネントの名前を `ProcModel` にします。
- Component → Edit を選択して、ESDL エディタを開きます。

  ここではもちろん外部テキストエディタも使用できます。使用方法は、チュートリアルの最初の部分に説明されています。



プロセスモデルを編集するには、まず必要なエレメントを追加してから、メソッド `derivatives()` および `nondirectOutputs()` を編集します。

プロセスモデルを編集する：

- ESDL エディタの **Continuous State** ボタンを使用して、2 つの連続ステートを作成します。
- これらのステートの名前を s および sp にします。

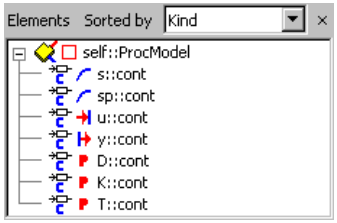
- **Input** ボタンをクリックして、入力を作成します。
- その入力の名前を u にします。

- **Output** ボタンをクリックして、出力を作成します。
- その出力の名前を y にします。

  どちらのエレメントも、cont 型です。

- **Parameter** ボタンをクリックして、パラメータを作成します。
- そのパラメータの名前を D にします。
- パラメータ K および T を作成します。

  このプロセスモデルの“Elements”リストは、以下のようになります。



- パラメータを以下のように設定します。

```
D = 0.4
K = 0.002
T = 0.05
```

- “Diagrams”リストから derivatives メソッドを選択し、そのコードを下図のように編集します。

derivatives []

```
s.ddt(sp);
sp.ddt((K*u - 2*D*T*sp - s)/(T*T));
```

この図は内部テキストエディタを使用した例です。

注記

微分方程式の解法については、『ASCET リファレンスガイド』の「連続系基本ブロック」の章の「基本事項」を参照してください。

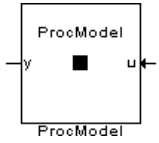
- “Elements”リストから nondirectOutputs メソッドを選択し、以下のテキストを入力します。

nondirectOutputs []

```
y = s;
```

- レイアウトエディタでレイアウトを調整します。

  プロセスモデルの場合、出力を左側、入力を右側に配置するのがよいでしょう。



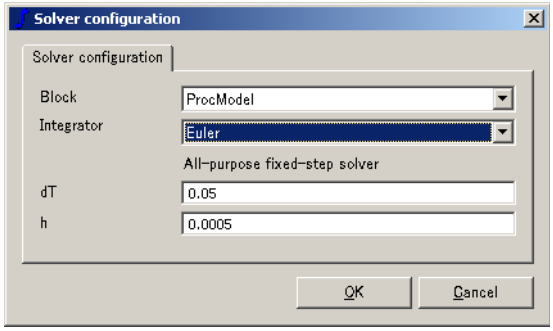
- Edit → Save を選択します。
- 
  コンポーネントマネージャの **Save** ボタンをクリックして、プロセスモデルを保存します。

これで、新しいモデルを用いて実験を開始する準備ができました。

モデルの実験を行う：

- ESDL エディタで **Component → Open Experiment** を選択して、実験環境を開きます。

- **Open CT Solver** ボタンをクリックして、“Solver Configuration”ダイアログボックスを開きます。
  現在のコンフィギュレーションが表示されます。



- dT の値を 0.005 に変更します。
- **OK** をクリックして設定を有効にします。
- データジェネレータを開き、入力 u のチャンネルを作成します。
- 以下の値をチャンネル u に設定します。

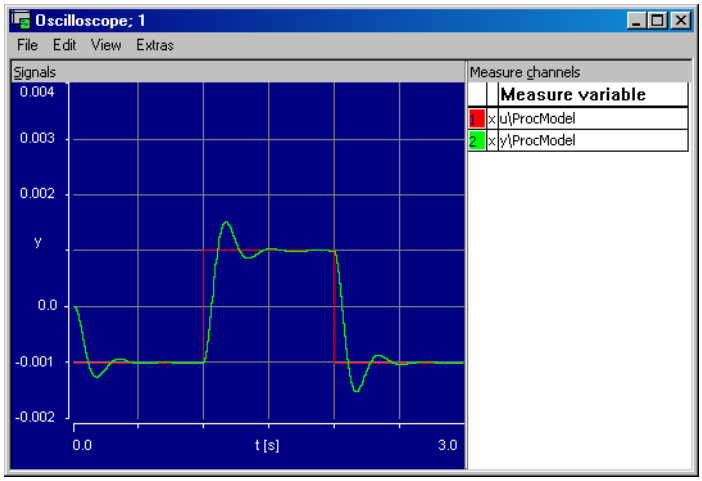
Mode：	pulse
Frequency：	0.5Hz
Phase：	0.0s
Offset：	-0.5
Amplitude：	1.0

- チャンネル u および y のオシロスコープウィンドウを開きます。
- オシロスコープの各測定チャンネルを、以下のように設定します。

	u	y
Min	-1	-0.002
Max	2	0.004
Extent	3.0	3.0

- **Save Environment** ボタンをクリックします。

- 実験を開始します。
  出力は下図のようになるはずです。



6.7.2 プロセスモデルを統合する

閉制御ループを作成するために、前に作成したコントローラプロジェクトにプロセスモデルを統合します。これまでのレッスンと同様に、モジュールを内包させ、オペレーティングシステムをセットアップしてグローバルエレメントを結び付けるという作業が必要です。

注記

作業を単純にするために、プロセスモデルを同じプロジェクトに追加します。この方法は、閉ループシミュレーションの早い段階でのテストにしばしば役立ちますが、プロセスモデルは組み込みシステムの構成要素ではないため、通常は、ネットワーク経由で各プロジェクト向けに配布されます。

プロセスモデルを内包させる：

- コンポーネントマネージャから `ControllerTest` 用のプロジェクトエディタを開きます。
  プロジェクトエディタで、コンポーネント `ProcModel` を“Components”リストに追加します。
- 
  プロジェクトエディタの“OS”タブを選択し、CT タスクのスケジューリングを定義します。

- タスク `simulate_CT1` を選択し、“Period” フィールドの値を 0.01（秒）に設定します。

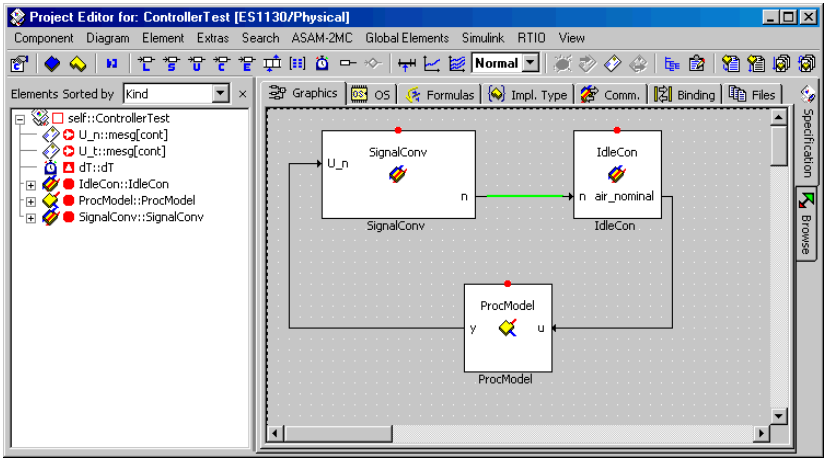
  コントローラとプロセスモデルが、共に同じ時間間隔で実行されます。

連続系ブロックとモジュールの結び付けを自動的に行うことはできないので、ブロックダイアグラムで明示的に接続する必要があります。

モジュールと連続系ブロックの結び付きを調整する：

- “Graphics” タブをクリックします。
- “Elements” リストから、3 つのコンポーネントをドラッグし、描画エリアにドロップします。
- モジュールのメッセージを、CT ブロックの対応する入出力に接続します。

  この例では、`ProcModel` の出力 `y` にグローバルメッセージ `U_n` を接続し、`ProcModel` の入力 `u` にグローバルメッセージ `air_nominal` をそれぞれ接続します。
- 各コンポーネントを右クリックして **Unconnected Ports** を選択し、未接続のポートがダイアグラムに表示されないようにします。



モジュール間通信のためのメッセージの結び付けは、自動的に行われます。同じ名前のメッセージ同士が結びつきます。

これでプロジェクトが完成し、実験を行えるようになりました。今度はオンライン実験を行うので、ASCET-RP がインストールされていることと、リアルタイムターゲット（ES1000 など）が接続されていることが必要となります。これらの条件が揃っていない場合は、これまでどおりオフライン実験を行ってください。

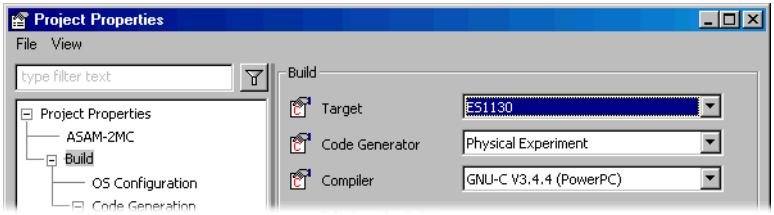
注記

オフライン実験を行う場合には、グローバルメッセージ U_n をデータジェネレータから必ず削除してください。

プロジェクトをオンライン実験用にセットアップする：

- **Specify Code Generation Options** ボタンをクリックします。
- “Settings”ダイアログボックスの“Build”タブで、ターゲットに ES1130、コンパイラに GNU-C (Power-PC) を選択します。

  これらのオプション設定により、ハードウェアとそれに対応するコード生成用コンパイラが指定されます。

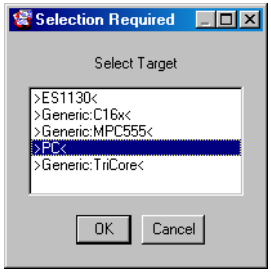


- **OK** をクリックして、ダイアログボックスを閉じます。

  Open Experiment for selected Experiment Target および **Reconnect to Experiment of selected Experiment Target** ボタンが有効になります。

- “OS”タブをクリックして、オペレーティングシステムエディタを起動します。

- 前に作成したスケジューリング設定をコピーするために、**Operating System → Copy From Target** を選択します。



- “Selection Required” ダイアログから PC を選択して **OK** をクリックします。

  これで、新しいターゲット用のプロジェクトに、以前の PC でのオフラインシミュレーション用に定義したものと同じスケジューリング設定が定義されました。

オンライン実験には、オフライン実験とは異なる点がいくつかあります。オンライン実験ではイベントジェネレータやデータジェネレータは用いません。イベントジェネレータは、オンライン実験用に生成されるオペレーティングシステムタスクのスケジューリングをオフライン実験でシミュレートするためのものです。

オンライン実験では、実験コードの実行（つまり OS の起動）と測定とを別々に開始します。このためツールバーにもそれぞれ専用のボタンが用意されています。これは、測定によって実験のリアルタイム動作が影響を受ける場合があるので、場合によっては測定を行わずに実験を行う必要があるためです。

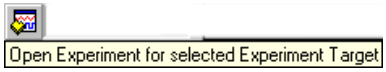
プロジェクトの実験をオンラインで行う：

- “Experiment Target” コンボボックスから `Online (RP)` を選択します。

  `Offline (RP)` はターゲット上でオフライン実験を行うためのモードです。

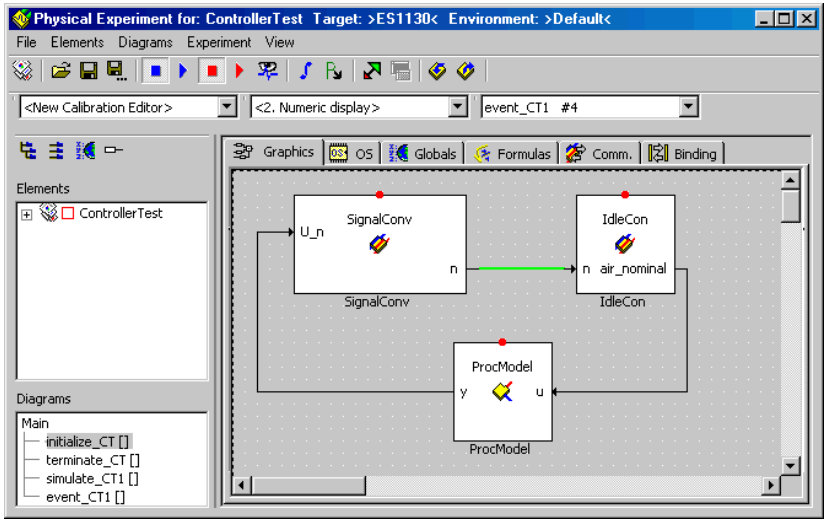
- **Component → Open Experiment** ボタンをクリックします。

または

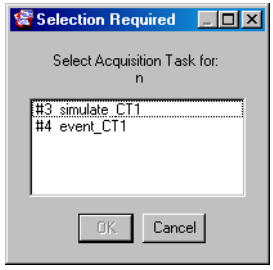


- **Open Experiment for selected Experiment Target** ボタンをクリックします。

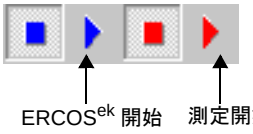
  実験用のコードが生成され、前に定義した環境と同じ環境で実験が開きます。



プロジェクトに複数のタスクが含まれている場合、各測定変数の値をどのタスクで取得するかを尋ねるダイアログボックスが開きます。



- “Selection Required”ダイアログボックスで、#3 simulate_CT1 というタスクを選択して **OK** をクリックします。
- `n` および `n_nominal` を既存のオシロスコープに設定し、それらの値の範囲を 0 ～ 2000 にします。
- 変数 `n_nominal`、`Ki` `Kp` 用の数値エディタを開きます。



- **Start ERCOS** ボタンをクリックしてから、**Start Measurement** ボタンをクリックします。

  実験が開始され、結果がオシロスコープに表示されます。`n` の値が急速に `n_nominal` に近づき、その状態が維持されます。
- `n_nominal` を数値エディタで修正します。

  `n_nominal` の値を変えると、それに応じて `n` の値が変わるはずです。
- パラメータ `Ki` および `Kp` を調整して、制御ループの動作を最適化することができます。

6.7.3 まとめ

このレッスンでは、ASCET で以下の作業を行いました。

- 連続系ブロックを作成し、定義する
- 連続系ブロックの実験を行う
- 連続系ブロックをプロジェクトに統合する
- 変数の結びつきを定義する
- 異なるターゲットに切り替える
- プロジェクトの実験をオンラインで行う

6.8 ステートマシン

ステートマシンは、有限数の明確なステート間を遷移するシステムをモデリングする際に有効な手段です。ASCET は、コンポーネントをステートマシンとして定義するための強力な機能を備えています。このレッスンでは、温度によるアイドリングエンジンの目標回転数の変化を表す、単純なステートマシンを定義しテストします。それから、そのステートマシンをプロジェクトに統合します。そして次のレッスンで階層ステートマシンを構築します。

エンジンが冷えている時には、高速アイドリングさせて回転を維持する必要があります。エンジンが暖まったら、燃料消費を減らすためにアイドリングの回転速度を下げます。このためのステートマシンは「エンジンが冷えている」および「エンジンが暖まっている」という 2 つのステートを持つ 2 段階制御を行います。

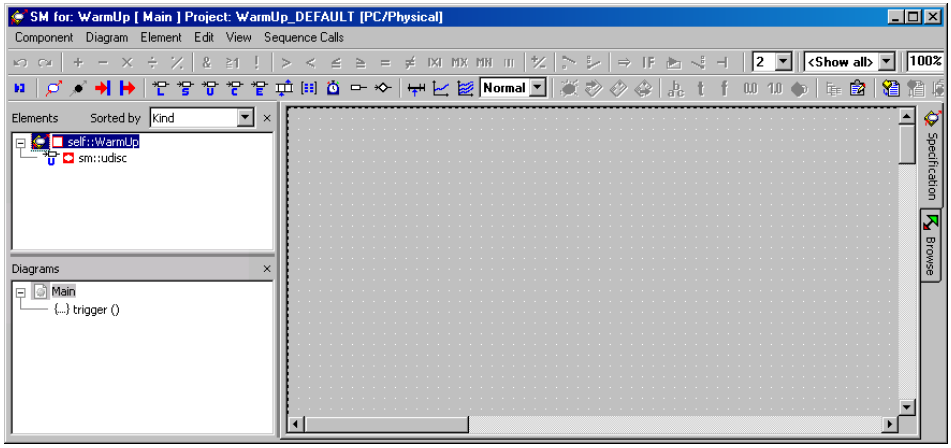
6.8.1 ステートマシンを定義する

ステートマシンはステート図と、アクションやコンディションの定義から構成されます。アクションやコンディションの定義により、さまざまなステートで、およびステートからステートへの遷移時に何を行うかを指定できます。これらはブロックダイアグラムと ESDL コードのどちらででも定義できます。

ステートマシンはブロックダイアグラムエディタで定義します。これ以外の方法としては、それぞれのステートやトランジションごとに開くテキストエディタにESDL コードを直接書き込む方法があります（注記：ここでは ESDL エディタは開きません）。ステートマシンには、他のコンポーネントとのデータ交換に使用される入力と出力があります。

ステートマシンを作成する：

- コンポーネントマネージャで、フォルダ Tutorial¥Lesson8 を作成します。
- **Insert → State Machine** を選択するか、または **Statemachine** ボタンをクリックして新しいステートマシンを作成します。
- 作成したステートマシンの名前を WarmUp にします。
- “1_Database”リスト内の新しいステートマシンの名前をダブルクリックして、ステートマシンエディタを開きます。



ステートマシンを作成する場合には、まずステートダイアグラムを定義してから、ステートやステートトランジションに関連付けるさまざまなアクションやコンディションを定義します。

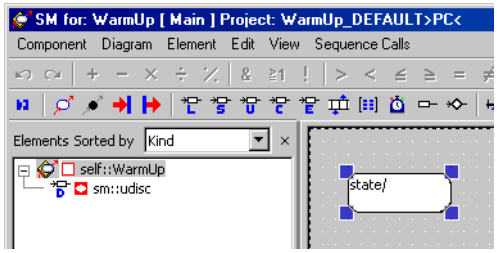
エンジンをコントロールするこのステートマシンには、「エンジンが冷えている」および「エンジンが暖まっている」という 2 つのステートがあります。

ステートダイアグラムを定義する：

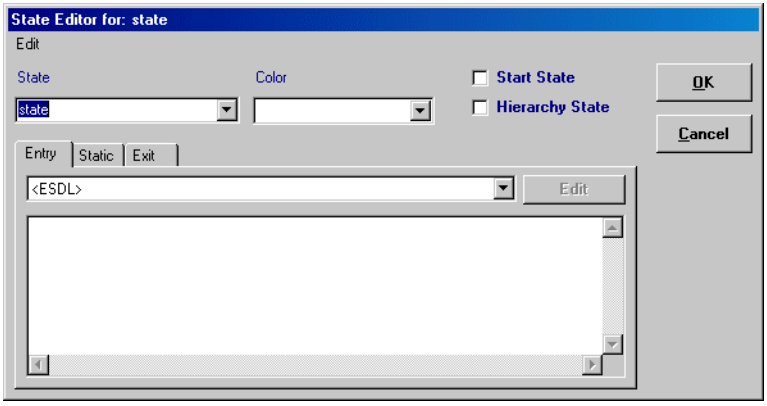
- **State** ボタンをクリックして、カーソルにステートアイテムをロードします。

- 描画エリア内の、ステートを配置したい位置をクリックします。

  クリックした位置にステートシンボルが表示されます。



- ステートシンボルをもう 1 つ作成し、描画エリア内の最初のステートの下に配置します。
- 先に作成した（上の）ステートシンボルを右クリックし、ショートカットメニューから **Edit State** を選択して、ステートエディタを開きます。



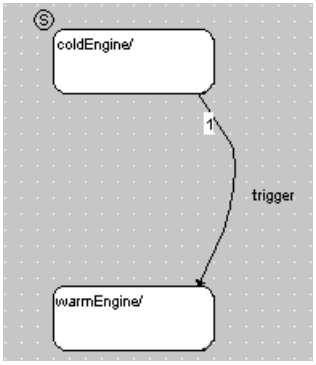
- “State” フィールドに `coldEngine` というステート名を入力します。
- **Start State** オプションをオンにして、このステートを、最初にステートマシンが起動されたときのステート（「開始ステート」）にします。

  ステートマシンには必ず開始ステートを 1 つ定義する必要があります。

- **OK** をクリックして、ステートエディタを閉じます。

  ステートの名前が、ステートシンボル内に表示されます。
- もう 1 つのステートシンボルの名前を `warmEngine` にします。
- 描画エリア内のシンボルが表示されていない位置を右クリックして、接続モードに切り替えます。
- `coldEngine`ステートシンボルの右半分をクリックして接続を開始し、次に `warmEngine` ステートシンボルの右半分をクリックして、両ステートを接続します。

  両ステートシンボルを結ぶ線が引かれます。この線は下向きの矢印になっていて、ステート間で起こり得るトランジションを表します。



- 下のステートシンボルの左半分から上のステートシンボルの左半分に向けて、もう 1 つのトランジションを作成します。
- **Diagram → Store to Cache** を選択し、作成したステートマシンを保存します。
- コンポーネントマネージャで **File → Save Database** を選択してデータベースを保存します。

ステートマシン構築の際に次に行うのは、そのインターフェースの定義です。温度値用の入力と、回転数用の出力が必要です。さらに、高温と低温、および回転数（rpm）を定義するパラメータが必要です。

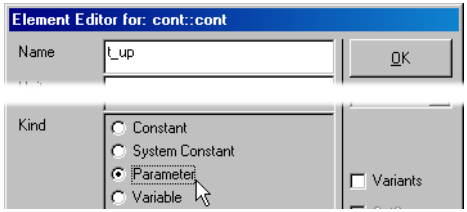
ステートマシンのインターフェースを定義する：

- **Input** ボタンをクリックして、入力を作成します。

- その入力の名前を t にします。

- **Output** ボタンをクリックして、出力を作成します。
- その出力の名前を n_nominal にします。

- **Continuous** ボタンをクリックして変数を作成します。
- エレメントエディタで、t_up という名前を入力し、**Parameter** オプションをオンにします。



これにより、作成された変数がパラメータになりました。

- 同じ方法で他のパラメータを作成します。
- 各パラメータの名前と値を以下のように設定します。

```
t_up = 70
t_down = 60
n_cold = 900
n_warm = 600
```

次に、両ステートと、その間のトランジションについて、アクションとコンディションを定義します。各ステートに定義できるアクションは以下の 3 種類です。

- そのステートに入るたびに実行される Entry アクション
- そのステートを離れるたびに実行される Exit アクション
- そのステート内に留まって、状態が変化しない時に実行される Static アクション

同様に、各トランジションには、トリガイベント、コンディション、優先度およびアクションを定義することができます。トリガ名とコンディション名は、トランジションの隣に表示されます。ステートマシンが作成されるとトリガが 1 つ、自動的に作成されます。

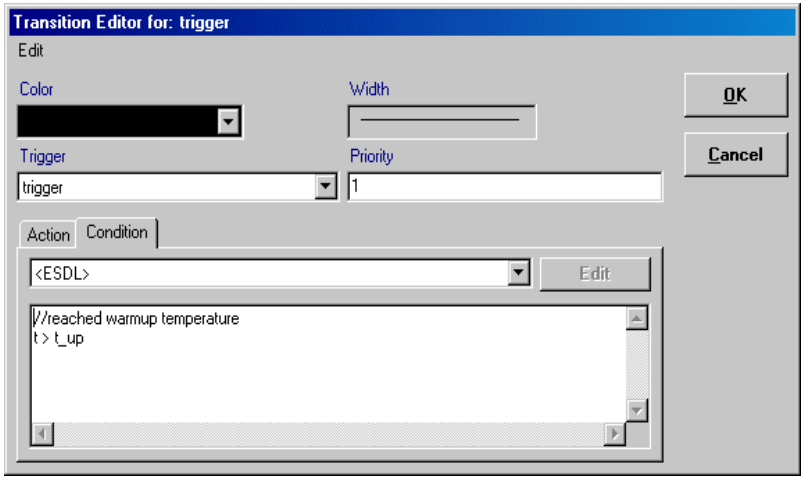
アクションとコンディションは、普通のダイアグラムか ESDL コードで定義されます。この例では、ESDL コードを使用します。

トリガのアクションとコンディションを定義する：

- coldEngine ステートから warmEngine に向かって接続されているトランジションを右クリックします。
- ショートカットメニューから **Edit Transition** を選択して、トランジションエディタを開きます。

  coldEngine から warmEngine へ遷移するための条件は、「実際の温度値 t が t_up より大きい」ということです。
- “Condition”タブのコンボボックスから <ESDL> を選択します。

  このコンボボックスではあらかじめ定義されているオプション設定を変更することができます。
- コンディションのコードペーンに、下図のコードを入力します。



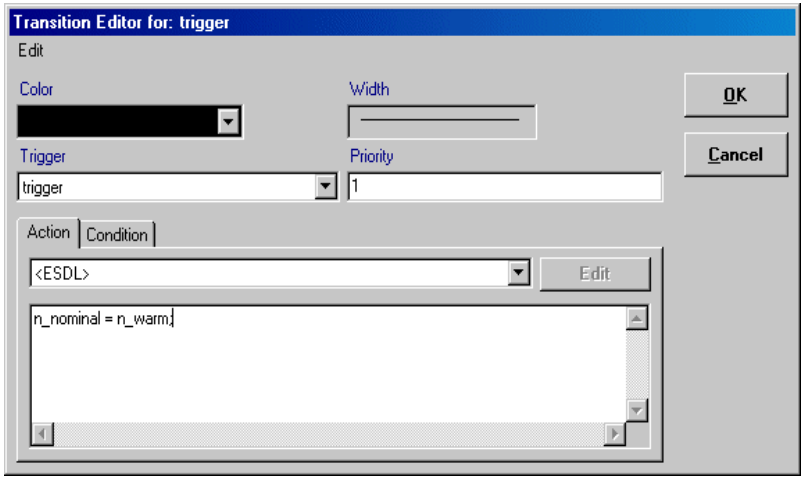
注記

トランジションエディタでは、コンディションの終わりにセミコロンを付けません。これは、通常の ESDL コードにおいてコンディションを括弧で囲んで記述する場合も同じです。

このコンディションの評価結果が `true` なら、エンジンのアイドリング速度が `n_warm` になります。

このコードはステートマシンダイアグラムに表示されます。この例では、トランジションのコンディションに別名（1 行目のコメント）を付けたので、それがダイアグラムに表示されます。

- “Action”タブでも `<ESDL>` を選択し、下図のコードを入力します。



- **OK** をクリックして、トランジションエディタを閉じます。

  ダイアグラムには、ステートマシンのコンディションとアクションが表示されます。

- `warmEngine` から `coldEngine` へのトランジションを編集するために、別のエディタを開きます。
- コンディションに `<ESDL>` を選択し、以下のコードを入力します。

```
t < t_down
```

このコンディションには別名（コメント）を付けなかったので、ダイアグラムにはこのコンディションのコード全体が表示されます。

- アクションに <ESDL> を選択し、下図のコードを入力します。

```
n_nominal = n_cold;
```

- エディタを閉じ、**Diagram → Store to Cache** を選択します。

アクションとコンディションは、ESDL の代わりに BDE で定義することもできます。ただしその際は、先にアクションとコンディション用のブロックダイアグラムを作成しておく必要があります。

アクションとコンディションのブロックダイアグラムを作成する：

- ステートマシンエディタで **Diagram → Add Diagram → Actions/Conditions BDE** を選択します。

  “Diagrams”ペーンに `ActionCondition_BDE` という名前のブロックダイアグラムが作成されます。

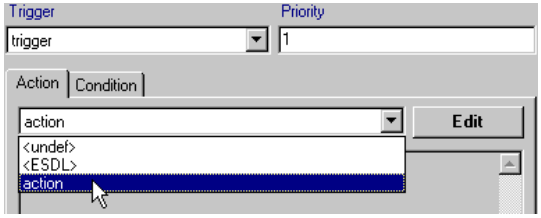
- このデフォルト名を確定します。

これでアクションとコンディションを定義することができるようになりました。

アクションとコンディションをブロックダイアグラムで定義する：

- “Diagrams”フィールドの `ActionCondition` ダイアグラムをクリックします。
- **Diagram → Add Action** と **Diagram → Add Condition** で、新しいアクションとコンディションを作成します。

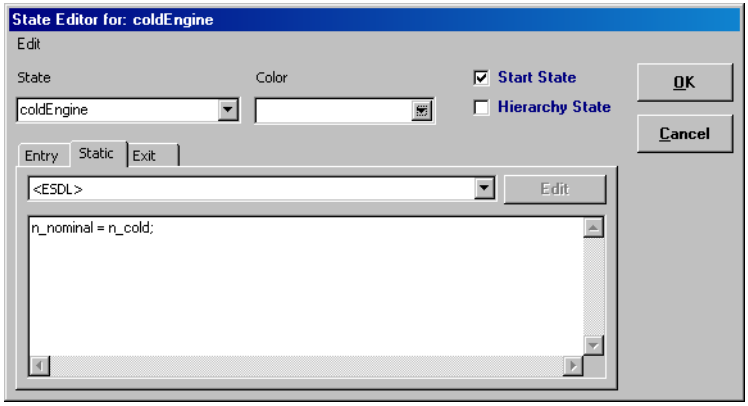
作成したアクションとコンディションは、“Transition Editor”ダイアログ内の各タブのコンボボックスで選択することができます。**Edit** ボタンをクリックすれば、直接 BDE を開いてグラフィック編集を行えます。



この時点で、出力 `n_nominal` の初期値はまだ定義されていません。パラメータとは異なり、この値を設定することはできないので、代わりに開始ステートである `coldEngine` のアクションを定義します。ただし開始ステートの Entry アクションは最初にステートマシンが起動された時には実行されないため、初期値の定義は Static アクション内で行う必要があります。

Entry アクションを定義する：

- 開始ステート `coldEngine` を右クリックします。
- ショートカットメニューから **Edit State** を選択して、ステートエディタを開きます。



- “Static”タブのコンボボックスから <ESDL> を選択して、Static アクションを定義します。

  ここでの設定は、コンポーネントマネージャの“Options”ウィンドウの“Defaults”タブ設定されているデフォルトの記述形式よりも優先されます。

- コードペーンに `n_nominal = n_cold;` と入力して、`n_nominal` の初期値を 900 にします。
- **OK** をクリックして、ステートエディタを閉じます。

これで、ステートマシンの定義が終わりました。この実験を開始する前に、ここであらかじめステートマシンの動作を理解しておいてください。

6.8.2 ステートマシンの動作

一般的に、標準コンポーネントの動作（機能）はそのグラフィック記述を見れば簡単に理解できますが、ステートマシンの場合はその動作が一見ではわかりにくい場合があります。本項では、前項の例を用いて、ステートマシンの原理について説明します。ステートマシンとその機能についての詳細は、『ASCET リファレンスガイド』の「ステートマシン」の章を参照してください。

ステートマシンの各ステートにはステート名、Entry アクション、Static アクション、および Exit アクションが定義されています。さらに、各ステートは他のステートからとの間にトランジション（状態遷移を表す接続線）を持ち、各トランジションには優先度、トリガ、アクションおよびコンディションが定義されています。

各ステートマシンにはそれぞれ 1 つの開始ステートが必要です。ステートマシンが初めて起動されると、開始ステートが有効になり、開始ステートのアクション（つまりステートマシンの初期処理）が実行された後、開始ステートから他のステートへのトランジションを発生させるコンディションが調べられます。この例では、このようなトランジションは 1 つしかなく、そのコンディションは `t > t_up` です。ここでは、入力値が `t_up` パラメータの値より大きいかどうかが調べられます。もしそうならコンディションは真（true）なので、トランジションが発生します。

パラメータ `t_up` および `t_down` は温度を規定します。エンジンがこれらの温度にならないと、目標回転速度を変更することができません。今の例では、エンジン温度が 70 度を超えたら、回転速度を 600rpm に下げることができます。その後、エンジン温度が 60 度を下回ると、目標速度を 900rpm にリセットしなければなりません。

トランジションが発生すると必ず、そのトランジションに定義されているトランジションアクションが実行されます。今の例では、`coldEngine` から `warmEngine` へのトランジションが発生すると実行されるトランジションアクション `n_nominal = n_warm` により、変数 `n_nominal` の値が 600 になります。反対に、トランジションアクション `n_nominal = n_cold` は `n_nominal` の値を 900 にします。トランジションが発生すると、離れようとするステートの Exit アクションと、入ろうとするステートの Entry アクションも実行されます。この例では、これらのアクションは定義されていないので、何も行われません。

ステートマシンが第 2 のステートに入ると、第 2 のステートから第 1 のステートへのコンディションが満たされるまでは、第 2 のステートに留まります。ステートマシンが 1 つのステートに留まっている間は、ステートマシンが起動されるたびに Static アクションが実行されます。ステートマシンを起動させるのは必ず外部のイベントで、1 回の起動によりステートマシンの 1 回のパスが開始されます。

ステートマシンの 1 回のパスは、まず現在のステートから他のステートへのトランジションについてのすべてのコンディションを調べることから始まります。コンディションは、その優先度の順に調べられます。あるコンディションが true なら、それに対応するトランジションが発生し、Exit アクション、トランジションアクションおよび Entry アクションが実行されます。最初に調べたコンディションが true だった場合、同じステートから他のステートへの他のトランジション（最初に調べたコンディションよりも優先度が低いコンディション）は調べられません。どのコンディションも true でない場合には、現在のステートのままになり、そのステートに留まるパスが実行されるたびに Static アクションが 1 回実行されます。

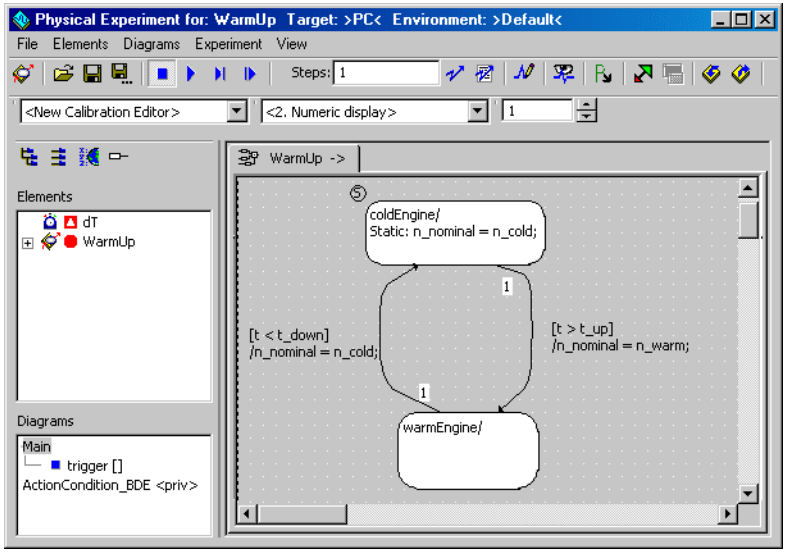
第 2 のトランジションのコンディションが true になると、つまり、入力値がしきい値よりも小さくなると、ステートマシンは第 1 のステートに戻ります。それから、入力値が再びしきい値より大きくなるまでは、そのステートのままになります（第 1 のステートには Static アクションが定義されていないので、何も行われません）。

6.8.3 ステートマシンの実験を行う

実験には、ステートマシンの場合も他のタイプのコンポーネントの場合と同様の機能がありますが、それ以外に、ステートマシンの実験を行う場合にだけ使用できるアニメーションという機能があります。これは、実験実行中にステートマシンダイアグラム内のカレントステート（現在アクティブになっているステート）を強調表示する機能です。

ステートマシンの実験を行う：

- ステートマシンエディタで **Component → Open Experiment** を選択して、実験環境を開きます。

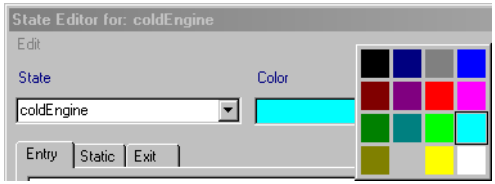


- ステートの 1 つを右クリックし、ショートカットメニューから **Animate States** を選択します。
- `trigger` イベントをイネーブルにします。
- データジェネレータで、変数 `t` のチャンネルを作成します。
- このチャンネルに周波数 1Hz、オフセット 70、振幅 20 の正弦波を割り当てます。
- `t` および `n_nominal` 用のオシロスコープウィンドウを開きます。

- **Start Offline Experiment** ボタンをクリックして、ステートマシンの実験を行います。

- 個々のステートの色を変えて表示を見やすくするために、**Exit to Component** ボタンで実験環境を終了し、ステートエディタを起動します。

- “Color”コンボボックスで色を選択します。



- 再度実験を行います。

実験が実行されると、n_nominal の値は、正弦波が対応する温度しきい値を上回ったり下回ったりするたびに変化します。適合システムを使用してしきい値を変更し、それによる出力の変化を調べることができます。また、ステートダイアグラムにおいては、カレントステートが強調表示されます。

6.8.4 ステートマシンをコントローラに統合する

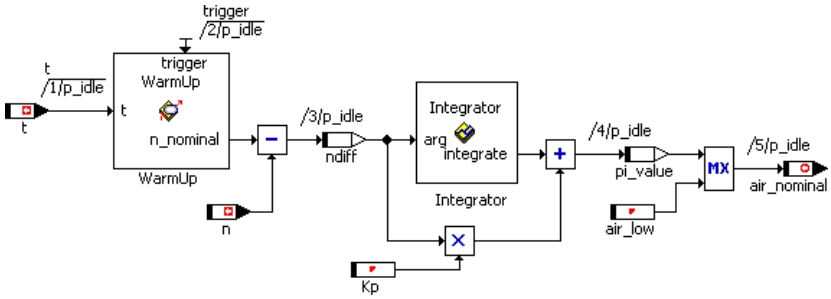
ASCET の他のすべてのコンポーネントと同様に、ステートマシンもやはり、別の任意のタイプのコンポーネントを構成する要素として用いることができます。では、ステートマシンをコントローラモジュールに統合し、回転速度をエンジン温度に合わせて調整できるようにしましょう。

ステートマシンを統合する：

- コンポーネントマネージャから、ブロックダイアグラムエディタでモジュール Lesson4¥IdleCon を開きます。
- ダイアグラムと“Elements”リストから、パラメータ n_nominal を削除します。

  ブロックダイアグラムにおいて、このパラメータの代わりにステートマシンを使用します。
- **Element → Add Item** を選択し、ステートマシンをコントローラの“Elements”リストに追加します。
- 受信メッセージを作成し、その名前を t にします。
- 削除した変数の代わりにコンポーネント WarmUp の出力を減算の演算子に接続し、同コンポーネントの入力を受信メッセージ t に接続します。

- ダイアグラムを下図のように調整します。すべてのアイテムが正しい順序で接続されるように、シーケンシングを調整します。



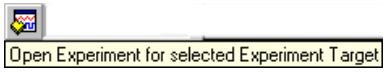
- ダイアグラムを保存し、コンポーネントマネージャで **Save** ボタンをクリックします。

修正後のコントローラをプロジェクト内で機能させるために、プロジェクトにも変更を加える必要があります。今度は、まだ使用されていなかった温度センサも統合します。

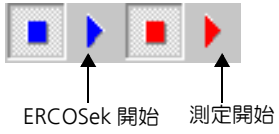
プロジェクトを変更する：

- プロジェクト `ControllerTest` 用のプロジェクトエディタを開きます。

- “OS”タブに切り替えます。
- プロセス`t_sampling`をタスク`Task10ms`に割り当てます。
- コマンド **Task → Move Up** を使用して、プロセス `t_sampling` をタスクの先頭に配置します。



- **Open Experiment for selected Experiment Target** ボタンをクリックします。
- 値 `U_t` 用に、もう 1 つのスカラ適合ウィンドウを開きます。
- 変数 `t` をオシロスコープに追加します。



- **Start ERCOS** ボタンをクリックします。
- **Start Measurement** ボタンをクリックします。
- 値 `U_t` を調整し、その影響を調べます。

  `t` の値が限界値（70 度）を超えると、ステートマシンは `n` の目標値を低い方の値 600 に切り替えます。温度が 60 度を下回ると（`U_t` を調整することによりシミュレートされます）、`n` の目標値は元の値 900 に戻ります。

6.8.5 まとめ

このレッスンでは、ASCET で以下の作業を行いました。

- ステートダイアグラムを作成する
- コンディション、アクション、およびトリガを作成し、ステートマシンに割り当てる
- ステートマシンの実験を行う
- ステートマシンを他のコンポーネントに統合する

6.9 階層ステートマシン

前のレッスンでステートマシンの機能について理解できたので、今度はやや複雑なシステムを構築してみます。このレッスンでは階層ステートマシンについて集中的に学習し、また、タイマなど ASCET に付属しているシステムライブラリとコンポーネントの使用法を学びます。

ASCET では、閉階層と開階層を用いてステートマシンを構築することができます。開階層の場合はその中に含まれるサブステートも図示され、閉階層の場合それらは省略されます。

交通信号制御システムを構築して、パラメータで指定できるタイミングを使用して交通信号の各フェーズへの移り変わりを実現してみましょう。この交通信号にはエラーステータスもあり、そのときには信号が点滅します。

6.9.1 ステートマシンを定義する

まず、必要なライブラリをインポートして準備します。

システムライブラリ `SystemLibETAS.exp` をインポートする：

- コンポーネントマネージャで **Import** ボタンをクリックします。

  “Select Import File”ダイアログボックスが開きます。
- “Import”フィールドの  ボタンを使用して、ASCET のインストールディレクトリ下のエクスポートディレクトリ（例：`C:¥etas¥ASCET5.2¥export`）に格納されている `SystemLibETAS.exp` というファイルを選択します。

  OK ボタンが有効になります。
- **OK** をクリックしてインポートを開始します。

  “Import”ダイアログボックスが開き、選択されたファイルに含まれるすべてのオブジェクトが一覧表示されます。

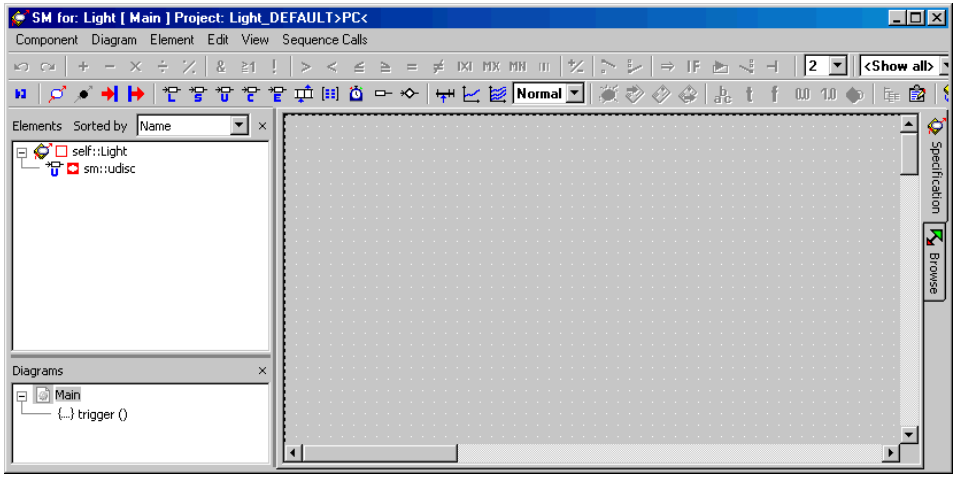
- **OK** ボタンをクリックします。

  ファイルがインポートされます。この処理には数分かかります。インポートが終了すると、インポートされたアイテムの一覧が“Import Items”ウィンドウに表示されます。

次に、交通信号を制御するために必要な 2 つのメインステート（`NormalMode` と `ErrorMode`）を定義します。

ステートマシンを作成する：

- コンポーネントマネージャで、フォルダ `Tutorial¥Lesson9` を作成します。
- **Insert → State Machine** を選択して、新しいステートマシンを作成し、その名前を `Light` にします。
- **Component → Edit** ボタンをクリックして、ステートマシンエディタを開きます。



ここから、交通信号を制御するステートマシンを定義します。

- 2 つの主要ステート `ErrorMode` および `NormalMode` を作成します。

次に、システムライブラリのタイマをプロジェクトに追加します。

タイマオブジェクトを追加する：

- **Element → Add Item** を選択します。

- “Select item”ダイアログボックスで、SystemLib_ETAS ライブラリの Counter_Timer フォルダにあるタイマオブジェクト、Timer を選択します。
- **OK** で確定します。

  オブジェクト Timer がステートマシンの“Element”リストに追加されます。

ステートダイアグラムを定義する：

- 必要なデータエレメントを以下のように定義します。
  - Logic 型の入力 error
  - 信号の 3 色を表す、Logic 型の 3 つの出力（yellow、green、red）
  - 信号のさまざまなフェーズを表す、Continuous 型の 4 つのパラメータ（BlinkTime、YellowTime、GreenTime、RedTime）

  依存パラメータについて理解を深めるために、緑色のフェーズの値だけを定義し、他のパラメータには緑色のフェーズの値に依存する値が設定されるようにします。

  ```
  RedTime = 2 * GreenTime
  YellowTime = GreenTime/3
  BlinkTime = YellowTime/10
  ```
- 次に、個々のパラメータの計算式と依存関係を定義します。
- このためには、パラメータ RedTime、YellowTime および BlinkTime について、エレメントエディタの“Dependency”の下にあるチェックボックス **Dependent** を選択します。

  エレメントエディタを起動するには、パラメータをダブルクリックするか、または **Edit** ショートカットメニューを使用します。
- **Formula** ボタンをクリックして、フォーミュラエディタを起動します。

- フォーミュラエディタを使用して、それぞれの依存パラメータについての計算式を定義します。最初に仮パラメータ x を作成し、それからフォーミュラペーンに計算式を入力します。

  ```
  Redtime : 2*x
  YellowTime : x/3
  BlinkTime : x/10
  ```

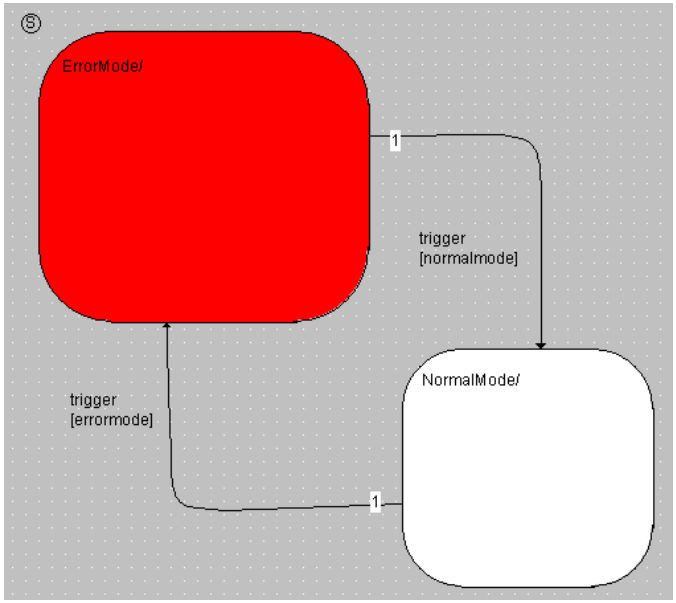
- フォーミュラエディタとエレメントエディタを閉じます。
- ショートカットメニュー **Edit Data** でディペンデンシーエディタを開きます。
- それぞれの依存パラメータの仮パラメータ x に対応するモデルパラメータを割り当てます。

  ```
  RedTime : x = GreenTime
  YellowTime : x = GreenTime
  BlinkTime : x = YellowTime
  ```

- データエレメントに有意義な値を与えます（例、`GreenTime = 5`）。
- ステート `ErrorMode` ステートエディタで開きます。

  ステートエディタを開くには、編集するステートをダブルクリックするか、**Edit State** ショートカットメニューを使用します。
- このステートを開始ステートとして定義し、色を赤にします。
- 両方のステートを拡大表示し、階層を挿入できるようにします。
- 2 つのステートの間にトランジションを作成します。
- ショートカットメニュー **Edit Transition** で、またはトランジションを示す曲線をダブルクリックして、トランジションエディタを開きます。
- トランジションダイアログボックスにコンディションを入力して、2 つのステートの間のトランジションを定義します。入力 `error` が `false`（つまり、エラーが発生していない）なら正常ス

テート NormalMode になり、エラーの場合には ErrorMode になるように、ESDL でコンディションを記述します。



- **Database → Store to Cache** を選択し、作業内容を保存します。
- コンポーネントマネージャで **File → Save Database** を選択し、データベースを保存します。
- これらのメインステートについて、実験を行ってみてください。

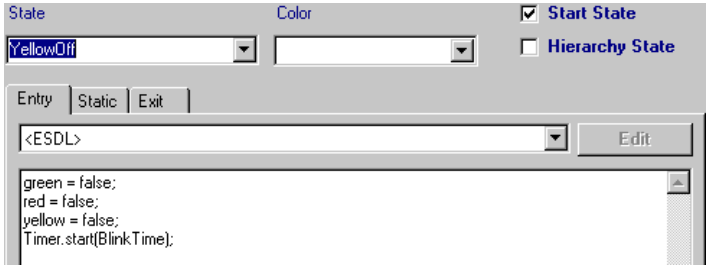
次に、交通信号制御システムに必要なサブステートを定義します。ここではまずエラーモード（ErrorMode ステート）における処理を定義します。このステートでは、黄色の点滅光が出力されます。このためには、YellowOff および YellowOn という 2 つのサブステートを定義し、この両者間の切り替えをタイマで行うようにします。YellowOn ステートでは出力 Yellow が true になり、YellowOff ステートでは false になります。

エラーモード用のサブステートを定義する：

- YellowOff および YellowOn というステートを作成し、それらをステート ErrorMode 内に配置します。

- `YellowOff` を開始ステートとして定義し、`YellowOn` の色を黄色にします。
- ステート `YellowOff` の対応をステートエディタで定義します。

  ステートエディタを開くには、**Edit State** ショートカットメニューを使用するか、このステートをダブルクリックします。
- Entry アクションを入力します。“Entry”タブのコンボボックスで ESDL を選択し、下図のコードを入力します。



- Static アクションを入力します。“Static”タブに以下のコードを入力します。

  ```
  Timer.compute();
  ```
- 次に、同じ方法で、`YellowOn` ステートについて定義します。

  Entry アクション：

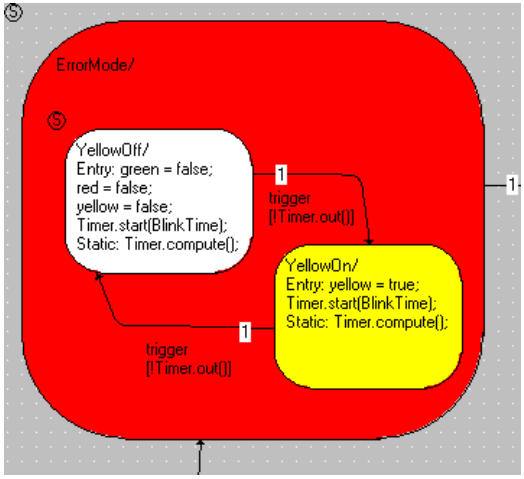
  ```
  yellow = true;
  Timer.start (BlinkTime);
  ```

  Static アクション：

  ```
  Timer.compute();
  ```

- 今度は、2 つのサブステートの間のトランジションを定義します。

  状態遷移の条件は、タイマがタイムアウトすること（`Timer.out() == false`）です。



つまり、`ErrorMode` ステートは `YellowOff` ステートで開始されます。このステートの Entry アクションで、黄色の点灯信号をオフにし、パラメータで指定される点滅時間を計測するタイマを起動します。このステートの Static アクションではタイマ関数 `compute()` が毎回起動され、タイマカウンタをデクリメントします。このカウンタ値が 0 になると、タイマ関数 `out()` がコード `false` を返し、トランジションのコンディションが真になります。そして、ステート `YellowOn` では、Entry アクションにより、黄色の点灯信号 `Yellow` がオンになります。

次に、正常動作時の動作を定義します。このためには、開始ステート `AllOff` を作成して `NormalMode` ステート内に配置します。そしてその Exit アクションで各色の点灯信号を定義されたステートに設定します。では、交通信号制御システムとして、どのように処理すればよいかを考えてみましょう。

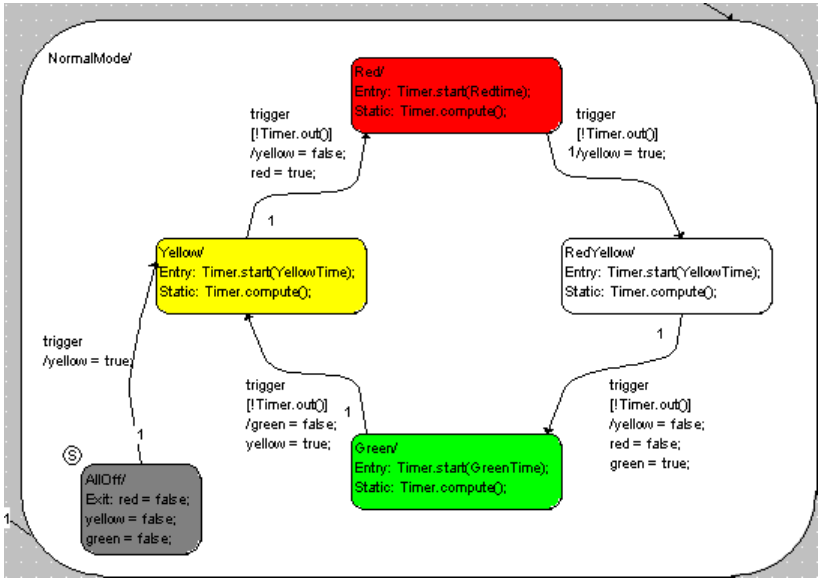
この例では、各色の点灯信号のオン／オフ切り替えを、今までのようにステートの Entry アクションにではなく、トランジションアクションに定義します。

正常動作時のサブステートを定義する：

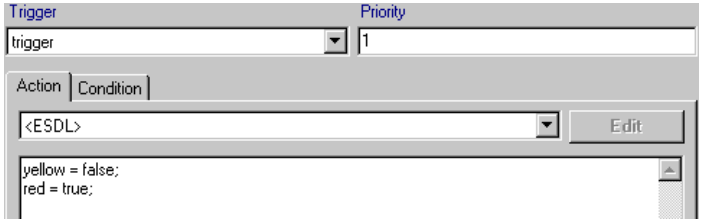
- ステート `AllOff`（開始ステート）、`Yellow`、`Red`、`RedYellow` および `Green` を作成して配置します。

- これらのステートの動作を定義します。これには、Entry アクションで各色用のタイマを起動し、Static アクションでタイマ処理（`Timer.compute()`）を行うようにします。
- ステートトランジションを定義し、トランジションアクション内にステートの動作を定義します。

  正常時は `AllOff` ステートから `Yellow` ステートに遷移し、他のトランジションは、対応するタイマのカウントダウンが終了した時に発生します。



- 各色の点灯信号用のアクションを、トランジションエディタの“Action”タブに入力します。



- トランジションエディタを閉じて、**Diagram → Store to Cache** を選択します。

これで、交通信号制御システムの定義が終わりました。この実験を始めるには、各色用タイマのパラメータに適切な値を入力する必要があります。

6.9.2 階層ステートマシンの実験を行う

階層ステートマシンの実験も、基本的なステートマシンの実験と同じ方法で行うことができます。実験では、アニメーション機能を起動することを忘れないでください。

ステートマシンの実験を行う：

- ステートマシンエディタで **Component → Open Experiment** を選択して、実験環境を開きます。
- ステートの 1 つを右クリックし、ショートカットメニューから **Animate States** を選択します。
- `trigger` イベントをイネーブルにします。

- **Start Offline Experiment** ボタンをクリックして、ステートマシンの実験を行います。

- `GreenTime` パラメータの値を変更し、さらに **Update Dependent Parameter** を使用して依存パラメータの値を更新して、実験を行ってください。
- `error` 入力を `true` にして、動作の変化を確認してください。

6.9.3 階層ステートマシンの動作

階層ステートマシンは、通常のステートマシンと同じように機能します。基本的に、階層ステートマシンはさまざまな動作すべてをグラフィック構造で表しているにすぎません。余裕のある方は、階層ステートマシンで定義した動作を、どのようにすれば階層なしで実現できるか考えてみましょう。

この交通信号の例では、2 つの階層ステートを使いました。システムは論理入力変数 `error` の値に応じて、2 つのステート `ErrorMode` と `NormalMode` を切り替えます。それぞれの動作は、これらのステートの中で定義されています。

これを理解するために、`ErrorMode` ステート内の処理に注目してみましょう。トリガが起動されるたびに、階層ステート `ErrorMode` から `NormalMode` へのコンディション（コンディション：`!error`）が調べられます。主要ステートのトランジションが必要ない場合には、サブステートのトランジション `YellowOff` から `YellowOn` へ、またはその逆のトランジションのコンディションが調べられ、必要なアクションが行われます。

`NormalMode` について考えてみると、トリガが呼び出されるたびに、まず `error` 入力が `true` かどうか、つまり遷移が必要かどうかが調べられ、この条件に該当しなかった場合にのみ、各サブステート（`AllOff`、`Yellow`、`Red`、`RedYellow`、`Green`）間のコンディションが調べられます。交通信号の例では、タイマがタイムアウトしたかどうかも調べられます。

以上の処理を明らかにするために、ステートダイアグラムから生成されるコードを見てみましょう。

生成されたコードを表示する：

- ステートマシンエディタで **Component → View Generated Code** を選択して、生成されたコードを表示します。

  コンポーネントのコードはテンポラリファイルに書き込まれ、Windows に登録されているアプリケーションで開かれます。

> 注記
>
> 生成されたコードを表示する際、拡張子 `*.c` および `*.h` のファイルに関連付けられているアプリケーションが、Windows のレジストリ内から検索されて開きます。

6.9.4 まとめ

このレッスンでは、ASCET で以下の作業を行いました。

- 階層ステートダイアグラムを作成する
- ステートアクションとトランジションアクションの動作を定義する
- モジュール、クラス、コンポーネントをインポートする
- ASCET ライブラリのシステムコンポーネントをインポートする
- 既存の Timer コンポーネントを使用する
- 依存パラメータを使用する
- 生成されたコードを表示する

7 用語集

この用語集では、ASCET のドキュメントで使用されている用語について解説します。各用語の一般的な意味よりも、ここでは、ASCET について使用される場合の意味について説明します。

用語はアルファベット順、あいうえお順に並んでいます。

7.1 略語集

ASAM-MCD

Assosiation for the **S**tandardization of **A**utomation and **M**easurement systems（オートメーションおよび測定システム標準化委員会）の、測定（**M**easurement）、適合（**C**alibration）および診断（**D**iagnosis）についてのワークグループ

ASCET

ECU ソフトウェア開発ツール

ASCET-MD

ASCET Modeling and **D**esign - ASCET モデリング／デザインツール

ASCET-RP

ASCET Rapid **P**rototyping - ASCET ラピッドプロトタイピングツール

ASCET-SE

ASCET Software **E**ngineering - ASCET ソフトウェアエンジニアリングツール

BDE

Block **D**iagram **E**ditor（ブロックダイアグラムエディタ）

CPU

Central **P**rocesing **U**nit（中央演算処理装置）

ECU

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

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

ESDL

Embedded **S**oftare **D**escription **L**anguage（組み込みソフトウェア記述言語）

ETK

独語：**E**mulator**t**ast**k**opf（Emulator Test Probe：エミュレータ・テストプローブ）

FPU

Floating **P**oint **U**nit（浮動小数点演算ユニット）

HTML

Hypertext **M**arkup **L**anguage（ハイパーテキストマークアップ言語）

INCA

Integrated **C**alibration and **A**cquisition Systems（統合計測・適合システム）

INTECRIO

ETAS の新しい製品ファミリ。さまざまなビヘイビアモデリングツールで作成されたコードを統合してコンフィギュレーション設定や実行コードの生成を行い、ラピッドプロトタイピング実験を行うための実験環境を提供します。

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）

RAM

Random **A**ccess **M**emory（ランダムアクセスメモリ）

ROM

Read **O**nly **M**emory（読み取り専用メモリ）

UML

Unified **M**odeling **L**anguage（統一モデリング言語）

XML

Extensible **M**arkup **L**anguage（拡張マークアップ言語）

7.2 その他の用語

ASAM-MCD-2MC ファイル

プロジェクト用のデータ交換に使用される ASCII 形式の標準フォーマットファイルで、測定および適合に必要なディスクリプションが含まれています。`*.a2l` という拡張子が付きます。

C コード

コンポーネントを記述する形式の 1 つで、内容はインプリメンテーションに依存します。

HEX ファイル

プログラムの 1 つのバージョンをハードウェア間で交換するためのファイル形式です。フォーマットは Intel HEX と Motorola S Record のいずれかです。

Intel Hex

プログラムの 1 つのバージョンをハードウェア間で交換するためのファイル形式です。

L1

ホストと、実験が行われているターゲットとの間でデータ転送を行うためのメッセージフォーマットです。データ転送は、たとえば測定ウィンドウでの値の表示のために行われます

Motorola-S-Record

プログラムの 1 つのバージョンをハードウェア間で交換するためのファイル形式です。

アイコン

ASCET コンポーネントの機能を視覚的に表すために使用されます。

アクション

ステートマシンの構成要素で、ステートマシンのステートまたはトランジションに関連付けて定義されます。一連の機能からなり、その実行はステートマシンによりトリガされます。

アプリケーションモード

ASCET のオペレーティングシステムの動作モードです。EEPROM プログラミングや暖機運転モード、通常モードなどシステムに想定されるさまざまな状態を表します。

イベント

オペレーティングシステムのアクション（タスクの起動など）の外部トリガとなるものです。

イベントジェネレータ

実験環境の一部を成すもので、オフライン実験において、タスク（メソッド／プロセス／タイムフレーム）起動のためのイベント生成の順序とタイミングを定義するために使用されます。

インターフェース

コンポーネントのインターフェースは、そのコンポーネントが他のコンポーネントとどのようにデータ交換を行うかを定義します。これは C 言語環境における .h ファイルにたとえられるものです。

インプリメンテーション

物理記述（モデル）から実行形式の固定小数点コードへの変換方法を定義するものです。モデルデータの変換に使用される線形変換式と上下限値とで構成されます。

インプリメンテーションキャスト

このエレメントを使用すると、連続する算術演算の中間値のインプリメンテーションを、そのエレメントの物理表記を変えることなく任意に変更することができます。

インプリメンテーション型

プロジェクト全体で使用できる複数のインプリメンテーションを「インプリメンテーション型」として定義しておき、それを必要に応じて各エレメントに個別に割り当てることができます。

ウィンドウエレメント

実験で使用される適合エレメントと測定エレメントを指す一般的な用語です。

エディタ

「適合ウィンドウ」を参照してください。

エレメント

コンポーネントを構成する要素で、データ（変数やパラメータ、またはコンポーネント内で使用される他のコンポーネント）の読み取りや書き込みを行います。

エレメントタイプ

エレメントのタイプには、変数、パラメータ、定数の 3 種類があります。変数の値は読み取りも書き込みも可能です。パラメータの値は読み取ることしかできませんが、実験中に適合（書き込みによる値の調整）を行うことができます。定数の値は常に読み取ることしかできず、実験中でも書き込みは行えません。

オシロスコープ

測定ウィンドウの一種で、実験中にデータの値をグラフ表示します。

オフライン実験

ASCET で生成されたコードが PC または実験ターゲット上で実行される環境ですが、リアルタイムな実行はサポートされません。ここでは主にシステムの機能記述についてのテストに重点が置かれます。

オペレーティングシステム

ASCET ソフトウェアシステムの起動や実行をスケジューリングするものです。また内部通信のためのサービス（メッセージ）やハードウェアの予約部分（リソース）へのアクセスもサポートします。ASCET のオペレーティングシステムは、ERCOS^{EK} というリアルタイムオペレーティングシステムをベースとしています。

オンライン実験

ASCET で生成されたコードが実験ターゲット上で、リアルタイムオペレーションシステムに設定された挙動に基づいてリアルタイムに実行されます。ここでは、スケジューリング等のリアルタイムな挙動 すが、リアルタイムな実行はサポートされません。ここでは主にシステムの機能記述についてのテストに重点が置かれます。

階層

階層ブロックは、ブロックダイアグラムによるグラフィック定義を階層化（構造化）するために使用します。

型（またはモデルデータ型）

ASCET モデルでは、`cont (continuous)`、`udisc (unsigned discrete)`、`sdisc (signed discrete)`、`log (logic)` といったさまざまな型の変数やパラメータを使用できます。`cont` は任意の値を持つことのできる物理量に使用され、`udisc` は正の整数値、`sdisc` は負の整数値に使用されます。また `log` は論理値（`true` または `false`）に使用されます。これらの型は、生成されるコードで使用される実装データ型とは異なります。加算や比較などさまざまな基本演算がそれぞれの型用に用意されています。インプリメンテーション情報によって、モデルデータ型が実装データ型に変換されます。

クラス

ASCET のコンポーネントの 1 つで、オブジェクト指向言語におけるクラスと同じものです。クラスの機能はメソッドにより定義されます。

グループ適合カーブ／マップ

1 つの座標ポイントディストリビューションを共有する複数の特性カーブ／マップを指し、総称して「グループテーブル」とも呼ばれます。各グループテーブルの出力値はそれぞれ異なります。座標ポイントのディストリビューションと個々のグループテーブルは、それぞれ独立したエレメントとして定義されます。

コード

コード（実行形式のコード）は、実際のアルゴリズムを含むプログラムで、データは含まれません。コードは、CPU によって実行されるプログラムの部分です。

コード生成

物理モデルから実行形式コードへの変換の第 1 ステップです。物理モデルが ANSI C コードに変換されます。C コードはコンパイラ（つまりターゲット）に依存するので、ターゲットごとに異なるコードが生成されます。

固定小数点コード

物理記述から生成されるコードの一種で、浮動小数点演算ユニットを持たないマイクロプロセッサ上で実行するためのものです。

コンディション

ステートマシンの制御フローを定義するためのものです。あるステートから別のステートへのトランジションを発生させるかどうかを決める論理値を返します。

コンテナ

プロジェクト、クラス、またはモジュールを保管しておくためのものです。モデルやデータベースの構築や、異なるデータベースアイテムを共通のバージョン管理下に置くために使用されます。

コンフィギュレーションダイアログボックス

測定／適合ウィンドウ、または各ウィンドウに表示される変数のコンフィギュレーションを行うために使用されるダイアログボックスです。

コンポーネント

ASCET における再利用可能な機能の基本単位です。コンポーネントはクラス、モジュール、またはステートマシンによって定義されます。各コンポーネントは、演算子で接続されて機能を形成する数々のエレメントで構成されます。

コンポーネントマネージャ

ASCET の作業環境設定を行ったり、ユーザーによって作成されデータベースに格納されたデータの管理を行うためのユーザーインターフェースです。

算術演算サービス

オンライン実験の環境設定が格納されます。測定／適合ウィンドウに関する情報（ウィンドウのサイズや位置、およびウィンドウに含まれるデータチャンネルの割り当てやサンプリングレート、表示形式等）や、イベントジェネレータ、データジェネレータ、データロガー等の設定が含まれます。

実験

コンポーネントやプロジェクトの機能をテストするために使用される実験環境の設定が定義されたものです。ここには、測定ウィンドウや適合ウィンドウに関する情報（ウィンドウのサイズや位置、およびその内容）や、イベントジェネレータ、データジェネレータ、データロガーについての設定が含まれます。実験は、オフライン（非リアルタイム）またはオンライン（リアルタイム）で行われ、バイパスまたはフルパス環境において物理プロセスを制御するシミュレーションが行われます。どの条件においても、ASCET で記述されたモデルから生成されたコードが使用されます。

実験環境

測定／適合作業を行うための操作環境です。

実装データ型

C 言語の基礎であるデータ型（`unsigned byte`(`uint8`)、`signed word`(`sint16`)、`float`）を指します。

スケジューリング設定

プロセスへのタスクの割り当て、およびオペレーティングシステムによるタスク起動に関する定義です。

スコープ

各エレメントは、ローカル（コンポーネント内でのみ使用可能）またはグローバル（プロジェクト内で定義されている）のスコープを持ちます。

ステート

ステートマシンを構成する要素です。ステートマシンは常にそのステートのうちの 1 つの状態（つまり、いずれか 1 つのステートがアクティブな状態）になっています。いずれか 1 つのステートが開始ステートとしてマークされていて、これがステートマシンの初期ステートになります。ステート同士はトランジション（遷移）を示す曲線で接続されています。各ステートには Entry アクション（そのステートに入ったときに実行されるアクション）、Static アクション（そのステートのまま変わらないときに実行されるアクション）、および Exit アクション（そのステートから出るときに実行されるアクション）が定義されます。

ステートマシン

ASCET のコンポーネントの 1 つです。その挙動は、トランジションで接続される複数のステートによって構成されるステートダイアグラムで記述されます。

測定値

実験中に画面上に表示される変数やパラメータの値です。値は、さまざまな測定ウィンドウ（オシロスコープや数値ディスプレイ）に表示することができます。

測定ウィンドウ

実験中に測定信号を表示する、ASCET の作業ウィンドウです。

測定チャンネルパラメータ

測定モジュールの個々のチャンネル用に設定されるパラメータです。

ターゲット

実験の対象となるハードウェアを指します。ターゲットには実験ターゲット（PC、トランスピュータ、PowerPC）とマイクロコントローラターゲット（ECU）があります。

ダイアグラム

コンポーネントをブロックダイアグラムまたはステートマシンによって定義するために使用される記述形式です。

タスク

オペレーティングシステムから起動される単位で、その中に複数のプロセスとその実行順が割り当てられます。タスクは、アプリケーションモード、起動用のトリガ、優先度、スケジューリングモード等の属性を持ちます。タスクが起動されて実行されると、そのタスクに割り当てられたプロセスが指定された順序で実行されます。

定数

ASCET モデルの実行中に値を変更することのできないエレメントです。

ディスクリプションファイル

ECU 内の特性値と測定変数の物理情報（名前、アドレス、変換式、ファンクション割り当てなど）を記述するファイルです。

ディストリビューション

1 つまたは複数の特性カーブ／マップのブレイクポイント（「サンプルポイント」とも呼ばれます）が定義されたものです。

ディメンション

基本エレメントのサイズを定義するものです。ディメンションの型は、スカラ（0 次元）、配列（1 次元）、特性カーブ／テーブルのいずれかです。

データ

プログラム用の変数の集合で、適合時に使用されます。

データジェネレータ

実験環境の一部を成すものです。実験対象のモデルの入力または変数をスティミュレートするために使用されます。

データセット

コンポーネントまたはプロジェクトのすべてのエレメントの初期データ、あるいはその参照が設定されています。

データベース

ASCET で生成されるすべての情報が格納されます。フォルダによる階層構造になっています。

コンポーネントマネージャ

データベースを作成し、格納されるデータを管理するための作業環境です。

データロガー

実験環境で測定されるデータを読み取り、後で分析できるようにディスクに保存する機能を持ちます。

適合

ASCET モデルの実行（実験）中に、パラメータの値（物理／インプリメンテーション）を調整することです。

適合ウィンドウ

パラメータの修正に使用する ASCET の作業ウィンドウです。

特性カーブ

2 次元のパラメータ（適合変数）です。

特性データ

マップ、カーブ、および特性値を表す一般的な用語です。（「パラメータ」も参照してください。）

特性値

1 次元のパラメータ（適合変数）です。

特性マップ

3 次元のパラメータ（適合変数）です。

トランジション

ステート間の遷移を表し、起こり得るステートの遷移について記述されたものです。各トランジションにはステートマシンのトリガの 1 つが割り当てられ、優先度、コンディション、アクションを持ちます。

コンディション

ステートマシンのフロー制御を定義するために使用されます。あるステートから別のステートへのトランジションを起こすかどうかを決める論理値を返します。

トリガ

タスクの実行、またはステートマシンの実行をトリガ（起動）するものです。

バイパス実験

ASCET がマイクロコントローラに直接接続され、マイクロコントローラソフトウェアの一部をシミュレートする実験です。

配列

基本スカラ型 `continuous` または `discrete` の静的な 1 次元リストで、基本スカラ型 `discrete` のインデックスを持ちます。

パラメータ

ASCET のモデル内での計算では値を変更できないエレメント（特性値、特性カーブ、および特性マップ）です。ただし実験中に適合させることができます。

引数

クラスメソッドへの入力です。その引数が属するメソッドの記述においてのみ使用でき、クラス内の他のメソッドでは使用できません。

フォルダ

ASCET データベースの階層構造を形成する単位で、この中に各アイテムが格納されます。

フルパス実験

ASCET が実験用マイクロコントローラに直接接続され、アプリケーション全体が ASCET によってシミュレートされる環境です。

プログラム

コードとデータから構成され、ECU の CPU により実行される 1 つの単位です。

プログラムバージョン

ECU プログラムが格納された HEX ファイルを指します。プログラムバージョンの一部が、データバージョンと共に 1 つのファイルとなります。

プロジェクト

組み込みソフトウェアシステム全体についての記述です。機能を定義するコンポーネント、オペレーティングシステムに関連した設定、および内部通信を定義するバインディングのしくみが記述されます。

プロセス

並行して実行される機能の単位で、オペレーティングシステムにより起動されます。プロセスはモジュール内に記述され、引数／入力や戻り値／出力を持ちません。

ブロックダイアグラム

コンポーネントをグラフィカルに記述するものです。さまざまなエレメント、演算子、および入力／引数や出力／戻り値が、方向性のある線で接続されます。1 つのブロックダイアグラムは複数のダイアグラムで構成されます。ブロックダイアグラムによる記述は、C コードによる記述とは異なり、物理記述です。

変換式

インプリメンテーション（実装情報）の一部で、データのモデルデータ型から実装データ型への変換方法を定義するものです。

変数

ASCET のモデルの実行中にモデルから値の読み取りや書き込みができるエレメントです。また適合作業において値を変更することもできます。

また広義では、パラメータ（特性データ）と測定信号を示す一般的な用語としても使用されます。

メソッド

オブジェクト指向プログラミングで使用されるクラスにおいて、そのクラスの機能の一部を記述するものです。メソッドは任意の数の引数と 1 つの戻り値を持ちます。

メッセージ

並行して実行されるプロセス間のデータ交換の整合性を保証するための、ASCET のリアルタイム言語構造体です。

モジュール

ASCET のコンポーネントの 1 つです。オペレーティングシステムによって起動される複数のプロセスが記述されています。モジュールを他のコンポーネント内のサブコンポーネントとして使用することはできません。

モニタ

実験中にエレメントの値をダイアグラム上に表示する機能です。

ユーザープロファイル

ユーザー固有のオプション設定です。

優先度

各タスクに対して数値で与えられる優先度で、この数値が大きいほど優先度が高くなり、優先的にスケジューリングされます。

リソース

組み込みシステムにおいて、排他的に使用される部分（タイマなど）をモデリングする際に使用されます。このような部分をアクセスするには、使用前にそれを予約（確保）して使用後に解放する、という処理が必要ですが、この一連の処理がリソースの概念によって行われます。

リテラル

コンポーネントの記述に使用されるもので、連続値や論理値として解釈される文字列を定義します。

レイアウト

コンポーネントは、入力／引数や出力／戻り値を表すピンや、タイムフレーム、メソッド、プロセスを表すグラフィック表現（レイアウト）を持ち、さらにそのレイアウトには、そのコンポーネントが他のコンポーネント内で使用される時のグラフィック表示に用いられるアイコンも含まれています。

8 参考資料

この項には、トラブルシューティング、ディレクトリ構成、および必要な参照ファイルについての情報が記載されています。また、キーボードコマンドの一覧も作業ウィンドウ別に掲載されています。

8.1 トラブルシューティングと ETAS へのお問い合わせ

ASCET の製品開発においては、プログラムの機能上の安全性が最も重要視されています。しかし万一ご使用中にエラーが発生した場合には、以下の情報を ETAS までお送りください。

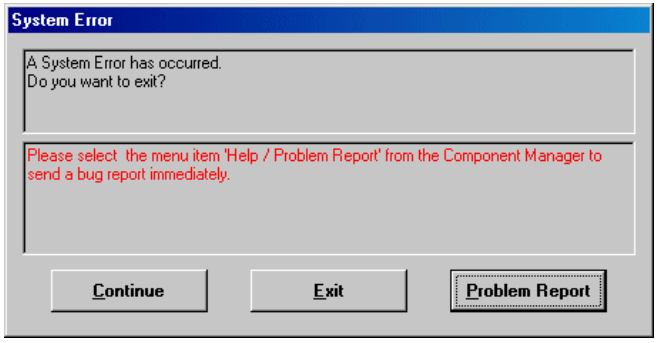
- エラーが発生したときにどのような作業をしていましたか？
- 発生したエラーはどのようなものでしたか？（関数エラー、システムエラー、システムのクラッシュなど）
- エラーが発生した時に、どのようなモデル（またはモデルエレメント）を編集していましたか？

注記

ASCET の品質をより高めるためには、ASCET の操作中にエラーが発生した際にユーザーの皆様からお送りいただくレポートは、非常に重要です。お手数ですが、できる限り以下の手順でレポートをお送りください。

サポート機能を使用すると、ASCET は自動的にログディレクトリの内容（すべての `*.log` ファイル）とテキスト情報を、`...¥ETAS¥LogFiles¥` サブディレクトリの `EtasLogFiles00.zip` というファイルに圧縮します。2 回目以降の ZIP ファイルには番号が付き、この番号は 1 から順に 19 までインクリメントされます。

重大なシステムエラーが発生すると、以下のようなウィンドウが開きます。



エラー発生時の操作：

1. **Problem Report** ボタン
   - **Problem Report** ボタンをクリックします。

     サポート機能が起動されます。
   - エラー内容を記述し、作成された ZIP ファイルを使用していたモデルと共に ETAS の LabCar ホットラインに送信してください。
1. **Exit** ボタン
   - **Exit** ボタンをクリックします。

     ASCET が終了します。保存されていなかった変更はすべて失われます。

     データを保存することを勧めるメッセージボックスが開いた場合、データの保存は行わずにメッセージボックスを閉じてください。
   - ASCET を再起動します。
2. **Continue** ボタン

> 注記
>
> **Continue** ボタンは、重要なコンフィギュレーションを保存する必要があるような場合に限り使用してください。そのまま作業を続行すると、さらにエラーが発生したり、不正なコンフィギュレーション設定が行われてしまう可能性があります。

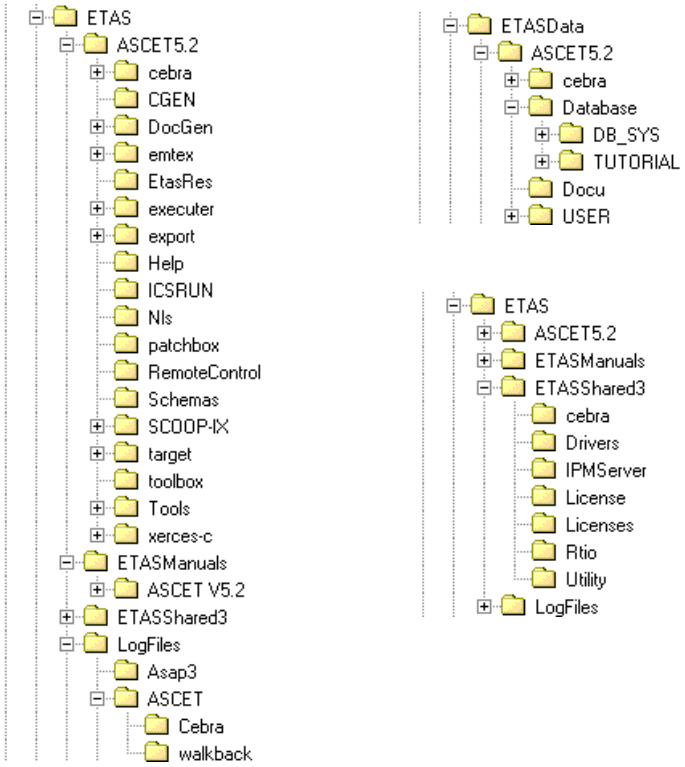
   - **Continue** ボタンをクリックします。

     ASCET は稼働し続け、エラー発生直前の状態に戻ります。
   - データを保存します。
   - ASCET を終了します。
   - ASCET を再起動します。

一般には、エラーが発生したらデータの保存を行わずにプログラムを終了し、再起動することをお勧めします。それにより、引き続いてより重大なエラーが発生することを回避できます。

8.2 ASCET のディレクトリ

ASCET をインストールすると、ユーザーがパス設定を変更しなかった場合に限り、以下のディレクトリ構成がインスタレーションディスクに作成されます。



8.2.1 デフォルトディレクトリ

- データベース
  `ETASData¥ASCET5.2¥Database`
- エクスポートファイル
  `ETAS¥ASCET5.2¥Export`
- 自動生成されるドキュメント
  `ETASData¥ASCET5.2¥Docu`

8.2.2 デフォルトディレクトリの変更

“Options”ダイアログボックスで、各種ファイルが格納されるデフォルトディレクトリを変更することができます。以下のように行ってください。

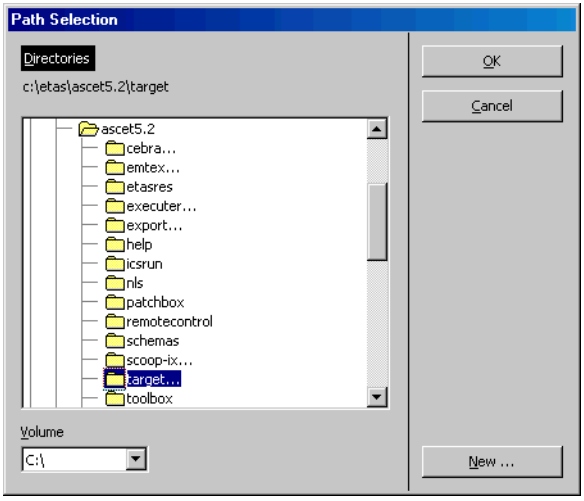
ファイルを格納するデフォルトディレクトリを変更する：

- コンポーネントマネージャから、メニューコマンド **Tools** → **Options** を選択します。

  ASCET のオプションダイアログボックスが開きます。データベースパスは“Options”ノードで変更し、エクスポートパスは“Export”ノード、ドキュメントパスは“Documentation”ノードでそれぞれ変更します。

- 変更したいパスの右側のボタンをクリックします。

  “Path selection”ダイアログボックスが開きます。



- デフォルトディレクトリとして使用したいディレクトリを指定します。
- **OK** をクリックします。

  指定されたディレクトリが“Options”ダイアログボックスに表示されます。
- 変更したいすべてのオプションについて、以上の手順を繰り返します。
- 設定が終了後、**OK** をクリックすると設定内容が確定され、**Cancel** をクリックすると取り消されます。

8.3 キーボード操作

8.3.1 一般的なキーボードコマンド

キー	機能
Alt + Tab	現在開いている他のアプリケーションに切り替える
Alt + Space	アプリケーションウィンドウのシステムメニューを開く
Alt + F4	アクティブなウィンドウを閉じる（コンポーネントマネージャの場合は ASCET を終了する）
< ↓ >, < ↑ >, < ← >, < → >	テーブルまたはリスト内のアイテムを選択するカーソルを移動、< ↓ > でリストフィールドが開きます。 （階層ビューにおいては < → > でフォルダが開き、< ← > でフォルダが閉じます）
Tab	フォーカス（強調表示）を、ウィンドウ内の次のアイテムまたはオプションに移動（<Shift> + <Tab> で逆方向に移動します）
Del	選択されたアイテムの削除
Space	テーブルまたはリスト内のアイテムの選択／解除
Esc	入力内容を取り消して入力モードを終了する
Shift	アイテムの複数選択を有効にする（Shift キーを押し下げたまま矢印キーでカーソルを移動させて範囲選択を行います）
Enter	入力内容を確定して入力モードを終了する（階層ビューにおいては < → >< ← > キーと同様の機能を持ちます）
Ctrl + A	全アイテムを選択する（例：リスト内の全アイテム）
Ctrl + C	データをクリップボードにコピーする
Ctrl + V	クリップボードのデータを貼り付ける
Ctrl + X	データを切り取ってクリップボードに貼り付ける
Ctrl + Y	最後の操作を取り消す（エディタでのみ有効）
Ctrl + Z	最後の操作を繰り返す（エディタでのみ有効）
Ctrl + F1	ホットキー（キーボードコマンド）の一覧を表示する
F10	メインメニューを開く

表 8-1 一般的なキーボードコマンド（個々のウィンドウにおいては機能が異なるコマンドもあります）

8.3.2 コンポーネントマネージャのキーボードコマンド

コンポーネントマネージャ では、一般的なキーボードコマンドに加えて以下のコマンドが使用できます。

キー	機能
Ctrl + N	新しいデータベースを作成する
Ctrl + O	データベースを開く
Ctrl + S	データベースを保存する
Ctrl + E	エクスポート機能を起動する
Ctrl + M	インポート機能を起動する
Alt + F4	コンポーネントマネージャを閉じて ASCET を終了する
F2	選択されたオブジェクトの名前を変更する
Ctrl + F	C コード（または ESDL）コンポーネント内の文字列を検索する
Ctrl + H	C コード（または ESDL）コンポーネント内の文字列を置換する
Ctrl + Q	データベース内のアイテムを検索する
F5	コンポーネントマネージャの表示内容を更新する
Insert	データベースのアイテムツリーにフォルダを挿入、またはコンテナ内にオブジェクトを挿入する
Return	選択されたデータベースアイテム用のエディタを開く
Alt + 1	“1 Database”ペーンをアクティブにする
Alt +2	“2 Comment”ペーンをアクティブにする
Alt +3	“3 Contents”ペーンをアクティブにする
Alt + F6	別のウィンドウをアクティブにする
Esc	選択部分の切り取り（<Ctrl> + <x>）をキャンセルする

8.3.3 モニタウィンドウのキーボードコマンド

ASCET *モニタウィンドウ* では、一般的なキーボードコマンドに加えて以下のコマンドが使用できます。

キー	機能
Ctrl + O	“Monitor”タブ上にログファイルを開く
Ctrl + S	“Monitor”タブ上のテキストをファイルに保存する
Ctrl + F	“Monitor”タブ上のテキストの検索／置換を行う

キー	機能
Ctrl + R	“Monitor”タブ上のテキストを消去する
Ctrl + +	モニタウィンドウのサイズを大きくする
Ctrl + -	モニタウィンドウのサイズを小さくする

8.3.4 各種エディタのキーボードコマンド

*ブロックダイアグラムエディタ*では、以下のキーコマンドが使用できます。

キー	機能
F2	選択されたオブジェクトの名前を変更する
Ctrl + →	次のシーケンスコールを選択する
Ctrl + ←	前のシーケンスコールを選択する

*ESDL エディタ*では、以下のキーコマンドが使用できます。

キー	機能
Ctrl + F	検索／置換
Ctrl + S	メソッド／プロセスを保存する

*AS エディタ*では、以下のキーコマンドが使用できます。

キー	機能
Ctrl + N	新しいファイルを作成する
Ctrl + O	ファイルを開く
Ctrl + S	ファイルを保存する

コンポーネントとプロジェクトの*インプリメンテーションエディタ*では、以下のキーコマンドが使用できます。

キー	機能
Alt + C, Alt + O	エディタウィンドウを閉じる
F2	選択されたデータセット／インプリメンテーションの名前を変更する

8.3.5 オフライン実験環境のキーボードコマンド

オフライン実験環境 では、以下のキーコマンドが使用できます。

キー	機能
Alt + F4	オフライン実験環境を閉じます
F10	メインメニューを開きます
Ctrl + C	エレメントの適合を行います
Ctrl + M	エレメントの測定を行います
Ctrl + S	エレメントに対するスティミュレーションを行います
Ctrl + L	データロガーでの選択されたエレメントの記録をアクティブにします
Ctrl + A	データロガーでの全エレメントの記録をアクティブにします
Ctrl + I	エレメントのインプリメンテーションを表示します
Ctrl + U	依存パラメータの値を更新します

8.3.6 測定／適合ウィンドウの一般的なキーボードコマンド

下記のキーボードコマンドは、*すべての測定／適合ウィンドウ*に共通して使用できます。各種ウィンドウに固有なキーボードコマンドは、以下に順にまとめられています。

キー	機能
Ctrl + H	アクティブウィンドウの変数の値を 16 進表示します
Ctrl + I	選択された変数に関する情報を表示します
Ctrl + P	アクティブウィンドウの変数の値を 10 進表示します
Ctrl + S	アクティブウィンドウ（3D グラフィックエディタを除く）の表示設定ダイアログボックスを開きます
Del	アクティブウィンドウ（グラフィックエディタと数値テーブルエディタを除く）から変数を削除します

表 8-2　すべての測定／適合ウィンドウで使用できるキーボードコマンド

適合ウィンドウ

*各種適合ウィンドウ*では、表 8-2 で示されたもの以外に、以下のキーボードコマンドを使用できます。

キー	機能
Ctrl + M	選択された値をインクリメントします（3D グラフィックエディタでは使用不可）
Ctrl + N	選択された値をデクリメントします（3D グラフィックエディタでは使用不可）
Ctrl + D	最後のアクションを繰り返します
Ctrl + U	最後のアクションを取り消します

以下のキーボードコマンドは、*数値エディタ*でのみ使用可能です。

キー	機能
Ctrl + R	変数の値を 2 進表示します
Ctrl + Z	変数の値を 10 進表示します
Ctrl + Page Up	選択された変数の表示位置を、ウィンドウ内で 1 つ上に移動します
Ctrl + Page Down	選択された変数の表示位置を、ウィンドウ内で 1 つ下に移動します

以下のキーボードコマンドは、*テーブルエディタ*でのみ使用可能です。

キー	機能
+	テーブルセルの値（Z 値）にオフセットを加えます
*	テーブルセルの値（Z 値）に係数を掛けます
=	テーブルセルの値（Z 値）に値を代入します
Ctrl + J	X 座標ポイントの値をデクリメントします（適合カーブ／マップの場合のみ）
Ctrl + K	X 座標ポイントの値をインクリメントします（適合カーブ／マップの場合のみ）
Ctrl + R	Y 座標ポイントの値をデクリメントします（適合カーブ／マップの場合のみ）

キー	機能
Ctrl + T	Y 座標ポイントの値をインクリメントします（適合カーブ／マップの場合のみ）
Ctrl + X	X 座標ポイントに値を代入します（適合カーブ／マップの場合のみ）
Ctrl + Y	Y 座標ポイントに値を代入します（適合カーブ／マップの場合のみ）

以下のキーボードコマンドは、*グラフィカルカーブエディタ*または *2D マップエディタ*でのみ使用可能です。

キー	機能
X	XZ 座標で表示します（2D マップエディタの場合のみ）
Y	YZ 座標で表示します（2D マップエディタの場合のみ）
Z	X（Y）座標と Z 座標を反転表示します
Ctrl + B	複数のカーブを選択する際に使用します（2D マップエディタの場合のみ）

以下のキーボードコマンドは、*3D マップエディタ*でのみ使用可能です。

キー	機能
← , →	マップを Z 座標軸を中心に回転します
↑ , ↓	マップを水平座標軸を中心に回転します
4, 6	マップを Z 座標軸を中心に回転します
8, 2	マップを水平座標軸を中心に回転します

測定ウィンドウ

*各種測定ウィンドウ*では、表 8-2 で示されたもの以外に、以下のキーボードコマンドを使用できます。

キー	機能
Ctrl + C	測定ウィンドウの現在の設定をクリップボードにコピーします
Ctrl + W	クリップボードにコピーされている設定を測定ウィンドウにコピーします

以下のキーボードコマンドは、*数値ディスプレイ*、*ビットディスプレイ*、および*横型／縦型棒グラフディスプレイ*でのみ使用できます。

キー	機能
Ctrl + Page Up	選択された変数の表示位置を、ウィンドウ内で 1 つ上（縦型棒グラフの場合は左）に移動します
Ctrl + Page Down	選択された変数の表示位置を、ウィンドウ内で 1 つ下（縦型棒グラフの場合は右）に移動します

以下のキーボードコマンドは、*数値ディスプレイ*でのみ使用できます。

キー	機能
Ctrl + R	変数の値を 2 進表示します
Ctrl + Z	変数の値を 10 進表示します

下記のキーボードコマンドは、*オシロスコープ*か *XY プロッタ*についてのみ使用できます。

キー	機能
Ctrl + A	選択された測定チャンネルの Y 軸のスケーリングを調整します
Ctrl + U	最後に行ったスケーリングの変更を取り消します
Ctrl + L	測定チャンネルリストの表示／非表示を切り替えます
Ctrl + X	選択された変数の表示／非表示を切り替えます
Ctrl + V	分析モードを開始／終了します
Ctrl + G	グリッドの表示／非表示を切り替えます
T	トリガイベントを発生させます
Page Down	“Measure Channel”または“Bit Channel”内の最後のチャンネルを選択します
Page Up	“Measure Channel”または“Bit Channel”内の先頭のチャンネルを選択します
← , →	選択された分析カーソルを 1 ステップだけ移動します（分析モードでのみ有効）
Ctrl + ← , Ctrl + →	選択された分析カーソルを指定のステップ分移動します（分析モードでのみ有効）

9 Windows XP ファイアウォールと ASCET

Windows XP には、パーソナルファイアウォールが組み込まれています。またそれ以外に、サードパーティ（Symantec、McAfee、Blackice）の各種パーソナルファイアウォールがインストールされていることがよくあります。

これらのパーソナルファイアウォールにより、ASCET-RP や ASCET-SE からイーサネットハードウェアへのアクセスがブロックされる場合あります。イーサネットで PC に接続されたハードウェアが、パラメータが正しく設定されているのに関わらずハードウェアの自動検索の際に検出されない場合、PC にインストールされているファイアウォールソフトウェアが原因となっている場合がよくあります。

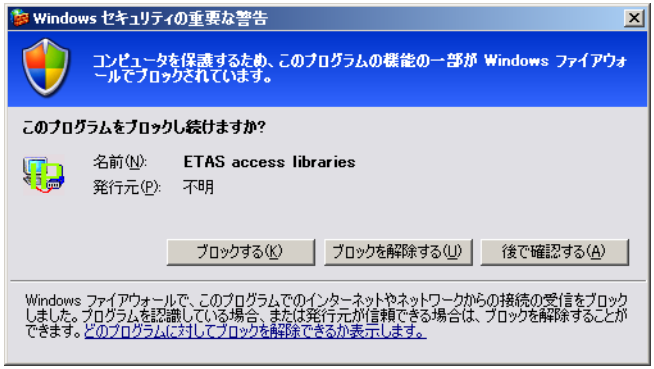
本章では、Windows XP（+Service Pack 2）の環境においてハードウェアアクセスが禁止されている場合に、Windows XP のファイアウォールを設定してこれを回避する方法を説明します。

Windows XP ファイアウォールが適切に設定されていないと、ETAS 製品において、以下のようなアクションが正しく実行されない可能性があります。

- ASCET
  - 実験を開く
  - 実験を再接続する
- HSP（ハードウェアサービスパック）
  - ハードウェアを検索する
  - ファームウェアアップデートを実行する
- INCA
  - ハードウェアを検索する
  - ハードウェアハードウェアコンフィギュレーションエディタを開く
  - 実験を開く

9.1 管理者権限を持つユーザーの場合

PC の管理者権限を持っているユーザーの場合、ETAS 製品がファイアウォールによってブロックされると、以下のようなダイアログボックスが開きます。



製品のブロックを解除する：

- “Windows セキュリティの重要な警告”ダイアログボックスで、**ブロックを解除する** をクリックします。

  以降、ファイアウォールによる ETAS 製品（例：ASCET）のブロックは行われなくなります。この設定は、プログラムや PC の再起動後も維持されます。

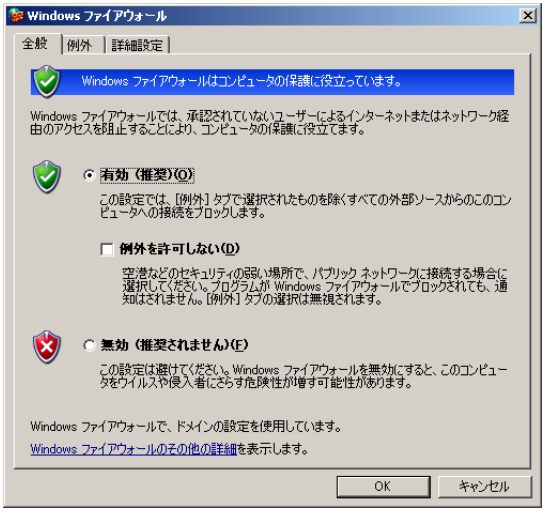
上記の“Windows セキュリティの重要な警告”ダイアログボックスが開く前に、前もって ETAS 製品のブロックを解除しておくこともできます。

ファイアウォールの設定を変更して製品のブロックを解除する：

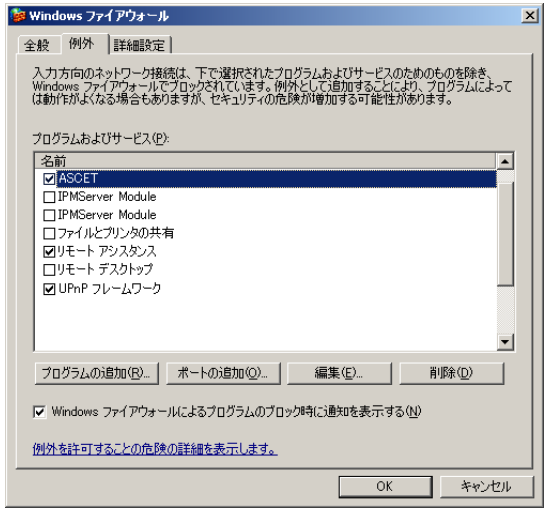
- Windows のスタートメニューから **設定 → コントロールパネル** を選択します。

- コントロールパネルから **Windows ファイアウォール** を選択します。

  “Windows ファイアウォール”ダイアログボックスが開きます。

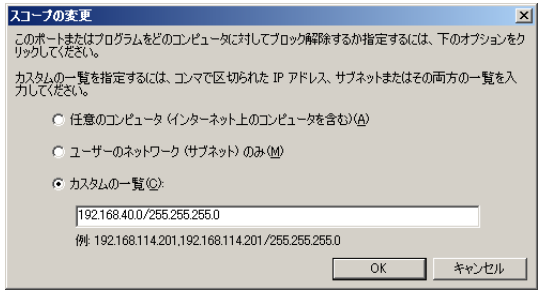


- “例外”タブを選択します。



  このタブには、ファイアウォールによるブロックから除外されているプログラムが一覧表示されます。

- **プログラムの追加** または **編集** ボタンをクリックして、新しいアイテム（プログラム）を追加するか、既存のアイテムを編集します。
- 以下のようにして、使用する ETAS 製品とそれに関連するサービスについて、正しく例外設定されていることを確認します。
  - 例外リストから該当するアイテムを選択して **編集** をクリックし、“プログラムの編集”ダイアログボックスを開きます。
  - **スコープの変更** をクリックし、“スコープの変更”ダイアログボックスを開きます。



- ETAS ハードウェアへのアクセスが行えるように、`192.168.40.xxx` という IP アドレスのうちのいずれかがブロック解除されていることを確認してください。
- **OK** をクリックして“スコープの変更”ダイアログボックスを閉じます。

- **OK** をクリックして“Windows ファイアウォール”ダイアログボックスを閉じます。

  以降、ファイアウォールによる ETAS 製品（例：ASCET）のブロックは行われなくなります。この設定は、プログラムや PC の再起動後も維持されます。

9.2 管理者権限を持たないユーザーの場合

権限（システム変更、書き込み、ローカルログオンなど）が制限されているユーザーの場合は、以下のように操作してください。

ETAS 製品を使用するユーザーは、所定のディレクトリ（`ETAS`、`ETASData`、ETAS の一時ディレクトリ）への“書き込み”の権利が必要です。それらの権利がない場合、ETAS 製品（ASCET など）を起動するとエラーメッセージが表示され、その後データベースが開きますが、正しい操作は行えません。これは、ETAS 製品の操作時にはデータベースファイルや `*.ini` ファイルの書き換えが必要なためです。

ASCET の場合、必ず管理者権限のあるユーザーがインストールを行い、その後、Windows XP ファイアウォールの例外リストに ETAS のプログラムを正しく登録してください。これが行われていないと、以下の事柄が生じます。

- 所定のアクション（221 ページ参照）を実行すると、以下のダイアログボックスが開きます。

製品のブロックを解除する（管理者権限のないユーザーの場合）：

- “Windows セキュリティの重要な警告”ダイアログボックスで、**このプログラムについてはこのメッセージを表示しない** をオンにします。
- **OK** をクリックしてダイアログボックスを閉じます。

  この後、管理者権限のあるユーザーが“Windows ファイアウォール”ダイアログボックスの“例外”タブで適切な設定を行い、ETAS 製品がハードウェアアクセスを行えるようにする必要があります。

9.3 お問い合わせ先

ご質問は、以下の ETAS ホットラインにおたずねください。

	電話	**E-Mail**
欧州（フランス、ベルギー、ルクセンブルグ、英国を除く）	+49–711–89661–311（ASCET） +49–711–89661–315（INCA）	ec.hotline@etas.de inca.hotline@etas.de
フランス、ベルギー、ルクセンブルグ	+33–1–5670–0235（ASCET） +33–1–5670–0234（INCA）	support.ascet@etas.fr support.inca@etas.fr
英国	+44–1283–546–512	support@etas-uk.net
日本	+81–45–222–0951（ASCET） +81–45–222–0950（INCA）	ec.hotline@etas.co.jp inca.hotline@etas.co.jp
韓国	+82(2)5747–101（ASCET） +82(2)5747–061（INCA）	ec.hotline@etas.co.kr inca.hotline@etas.co.kr
米国	+1–888-ETASINC（382–7462）	support@etasinc.com

10 お問い合わせ先

製品に関するご質問等は、各地域の ETAS 支社までお問い合わせください。

ETAS 本社

ETAS GmbH

Borsigstrasse 14
70469 Stuttgart
Germany

Phone: +49 711 8 96 61-0
Fax: +49 711 8 96 61-105
E-mail: sales@etas.de
WWW: www.etasgroup.com

北米

ETAS Inc.

3021 Miller Road
Ann Arbor, MI 48103
USA

Phone: +1 (888) ETAS INC
Fax: +1 (734) 997-9449
E-mail sales@etas.us
WWW: www.etasgroup.com

日本

イータス株式会社

〒 220-6217
神奈川県横浜市西区
みなとみらい 2-3-5
クイーンズタワー C 17F

Phone: (045) 222-0900
Fax: (045) 222-0956
E-mail: sales@etas.co.jp
WWW: www.etasgroup.com

英国

ETAS Ltd.

Studio 3, Waterside Court
Third Avenue, Centrum 100
Burton-upon-Trent
Staffordshire DE14 2WQ
UK

Phone: +44 1283 - 54 65 12
Fax: +44 1283 - 54 87 67
E-mail sales@etas-uk.net
WWW: www.etasgroup.com

フランス

ETAS S.A.S

1, place des Etats-Unis	Phone:	+33 1 56 70 00 50
SILIC 310	Fax:	+33 1 56 70 00 51
94588 Rungis Cedex	E-mail	sales@etas.fr
France	WWW:	www.etasgroup.com

韓国

ETAS Korea Co., Ltd.

4F, 705 Bldg. 70-5	Phone:	+82 2 57 47-016
Yangjae-dong, Seocho-gu	Fax:	+82 2 57 47-120
Seoul 137-889	E-mail	sales@etas.co.kr
Korea	WWW:	www.etasgroup.com

中国

ETAS (Shanghai) Co., Ltd.

2404, Bank of China Tower	Phone:	+86 21 5037 2220
200 Yincheng Road Central	Fax:	+86 21 5037 2221
Shanghai 200120	E-mail	sales.cn@etasgroup.com
P.R. China	WWW:	www.etasgroup.com

インド

ETAS Automotive India Pvt. Ltd.

No. 690, Gold Hill Square, 12F	Phone:	+91 80 4191 2585
Hosur Road, Bommanahalli	Fax:	+91 80 4191 2586
Bangalore, 560 068	E-mail	sales.in@etasgroup.com
India	WWW:	www.etasgroup.com

索引