ASCET MATLAB Integration Package V5.2

ASCET MATLAB® 統合パッケージ

ユーザーズガイド

著作権について

Document EC012101 R5.2.1 JP

目次

Simulink® インターフェース

1 はじめに

MATLAB® Integration Package（**MIP**）は、ETAS の組み込み制御システム開発ツール「ASCET」のアドオン製品で、MATLAB とのモデルデータ交換をサポートする以下の 2 つの部分から構成されています。

- **MATLAB® インターフェース** は、シミュレーションデータの交換や MATLAB 関数のリモート呼び出しの機能を持っています。MATLAB エンジンによってサポートされる MATLAB ワークスペースとのインターフェース機能は、オフライン実験において利用することができます。
- **Simulink® インターフェース** によって、Simulink や Stateflow® で作成されたモデルが ASCET にインポートされ、オフライン／オンライン実験において ETAS の実験ハードウェア上で実行することが可能となります。Simulink モデルのインポートは、Real-Time Workshop® を使用してモデルから C コードを生成すれば、それらが ASCET にモジュールとして自動的に組み込まれます。モデルを実行するために必要なタスクとプロセスは、ASCET 内で自動的に生成されます。

ASCET:

ASCET は、組み込み制御システム設計のための開発ツールです。主に自動車制御の分野で活用され、オブジェクト指向によるデザインコンセプトを持ち、シミュレーションや、ターゲットシステム用のコード生成に至るまで、リアルタイムなソフトウェアシステム用の開発プロセスをトータルにサポートします。Hardware-In-the-Loop（HIL）による検証が可能なため、システムの初期段階において制御アルゴリズム（開ループあるいは閉ループアルゴリズム）をテストすることが可能となります。これにより、新しいシステムの開発や既存のシステムの変更作業において、時間とコストの大幅な削減が実現されます。

MATLAB®/Simulink®:

MATLAB/Simulink プログラムパッケージは、2 つのコンポーネントから構成されます。ひとつは数学処理エンジンとプログラミング環境（MATLAB）で、もうひとつは数学モデルのグラフィック記述に関する処理を行う部分（Simulink）です。

Simulink の基本ブロックである MATLAB は、数学的問題に関する数値演算用に設計されていて、主にマトリックスやベクトル演算が基本となっています。これらのマトリックスやベクトル演算を制御するため、MATLAB の製造元である MathWorks 社によって、関数と問題解析アルゴリズムによって機能拡張を行うための言語が MATLAB に組み込まれました。MATLAB は科学や工業の分野で幅広く使用されており、システム解析や制御エンジニアリング（線形／非線形システム、ファジー制御など）の分野において、長年にわたり数多くの拡張機能が開発されてきました。

Simulink は、ラプラス変換と Z 変換を利用した数学的モデルのグラフィック記述をサポートします。MathWorks 社によって統合された基本ブロックライブラリを使用して、たとえば基本ブロックのようなものを接続していくことによって、データフローチャートの形式でモデルを構築することができます。Simulink のオープンなアーキテクチャにより、カスタマイズしたブロックを追加して基本ブロックを拡張することも可能です。非線形ダイナミックシステムのシミュレーションは、MATLAB により、数値計算機能と統合モデルを用いて実行されます。

Simulink には、モデルパラメータ用の変数をユーザー定義する機能があり、この変数の値は MATLAB 内に保存されます。このように、変数の値をモデル記述とは別に MATLAB（または `*.m` ファイル）内に保存することにより、モデル記述とパラメータ値との分離が可能となります。そのため、前もってさまざまな種類のデータセットを用意しておき、そのうちのいずれかを MATLAB 内にロードすることによって、さまざまなデータセットをモデルに代入してシミュレーションすることが可能となります。

1.1 ユーザーズガイドについての注意点

このユーザーズガイドは、MATLAB/Simulink の数学モデルのモデリング機能を ASCET と共に活用することを目標とする、組み込み制御システムの開発者を対象としています。そのため、ここに説明されている内容をご理解いただくには、MATLAB/Simulink や ASCET についての基本的な知識が必要です。これらの知識をお持ちでない方は、最初に各ツールのユーザーズガイドやチュートリアルをお読みいただくことをお勧めします。

1.2 ユーザーズガイドの構成

このマニュアルは、以下のように構成されています。

MATLAB インターフェースについては、第 2 章から第 6 章で説明されています。第 2 章で概要、第 3 章で機能と基本操作を説明し、第 4 章でモデリングや実験を行う際に MATLAB インターフェースを使用する方法について説明します。また API インターフェースについては第 5 章に、トラブルシューティングについてのヒントの一覧は第 6 章にまとめられています。

Simulink インターフェースのシステムアーキテクチャ、表記方法、およびモデルのインポートの操作に関しては、第 7 章で説明します。ここにはトラブルシューティングについてのヒントの一覧も含まれています。

なお製品の最新バージョンに関する詳細な情報は、リリースノートをお読みください。

1.3 表記上の規約規則

本書の表記は、以下の規約に従っています。

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。

表記例	説明
“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。

MATLAB Integration Package

MATLAB® インターフェース

2 MATLAB インターフェース ー 概要

2.1 MATLAB インターフェースとは

MATLAB インターフェースは、ASCET と MATLAB 間のデータ交換を可能にします。また、MATLAB 関数や MATLAB スクリプト（MATLAB の内部関数および M ファイル）を ASCET 環境から呼び出すこともできます。さらには、ASCET と Simulink モデル間の相互シミュレーションも、この MATLAB インターフェースによって実現できます。

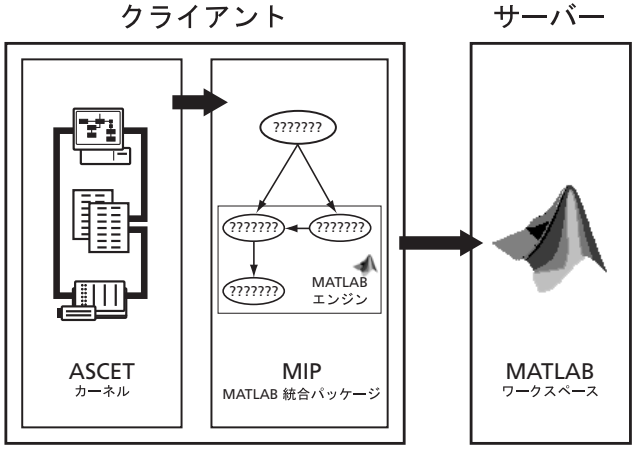
2.2 ASCET への MATLAB 関数の統合

MATLAB とのインターフェースにより、ASCET 開発環境での制御設計の解析工程における相互作業が可能となります。また、MATLAB のすべての関数を ASCET（データのポストプロセッシング、視覚化、ツールボックス等）と Simulink/Stateflow（モデル統合）で利用することができます。

3 MATLAB インターフェース ー 機能

3.1 MATLAB API エンジンを利用した MATLAB - ASCET インターフェース

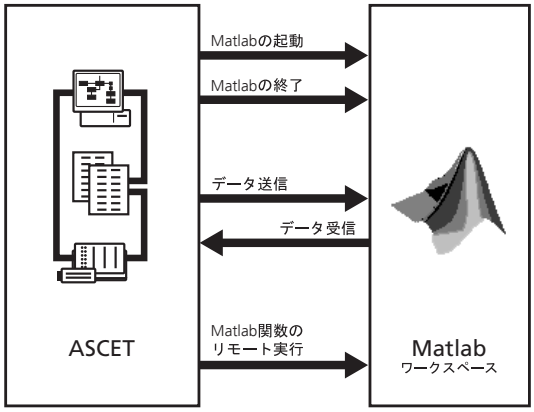
MIP は、MATLAB のプロセス間通信インターフェースである「MATLAB エンジン」を基本としています。MATLAB エンジン は MATLAB と他のアプリケーションの間でデータを交換するための基本機能を持っています。MATLAB はリモート操作によって起動されてサーバーとして機能し、リクエストを出すアプリケーション側がクライアントとなります。



MATLAB *エンジン* を使って、以下の処理が行えます。

- MATLAB プロセスのリモート起動と通信接続の確立
- MATLAB へのクライアントデータの送信
- クライアント環境での MATLAB データの受信
- MATLAB 関数のリモート実行
- 通常 MATLAB ワークスペース に表示されるタスクをテキストバッファに迂回する処理
- MATLAB プロセスの終了と通信接続の切断

また、MIP は、接続の確立と切断、メモリの取得と解放、エラーの監視と対処、および MATLAB ワークスペース 内の MATLAB データ構造体の生成（ASCET から行う MATLAB 変数の定義）といった内部管理も分担します。これにより、ユーザーや開発者が利用できる関数は C ソースコードで容易に組み込むことができるため、内部プロセスに関しての深い知識は特に必要ありません。



MIP には以下の C 関数が含まれています。

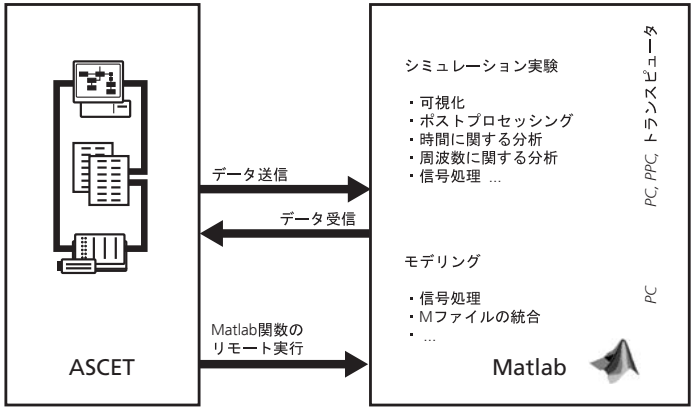
- MATLAB および通信パスを開く／閉じる：
  - `MIOpen`,
  - `MIClose`
- 実数データの送受信：
  - `MIPutMatrix, MIGetMatrix`
  - `MIPutVector, MIGetVector,`
  - `MIPutScalar, MIGetScalar,`
  - `MIGetResult`
- 複素数データの送受信：
  - `MIPutComplexMatrix, MIGetComplexMatrix,`
  - `MIPutComplexVector, MIGetComplexVector,`
  - `MIPutComplexScalar, MIGetComplexScalar,`
  - `MIGetComplexResult`
- 文字列変数の受信：
  - `MIGetString`
- MATLAB インストラクション／関数／スクリプトの実行：
  - `MIExecute`,
  - `MIExecuteAndEcho`,
  - `MIExecuteWithourErrCheck`

- MATLAB データ構造体のチェック：
  - `MIIsReal`,
  - `MIIsComplex`,
  - `MIisEmpty`,
  - `MIIsString`,
  - `MIGetNRows`,
  - `MIGetNCols`

MIP 関数については、第 5 章「Matlab インターフェース ー リファレンス」を参照してください。なおその中には、通常は直接呼び出されることはなく、内部的にのみ使用されるものも含まれています。

3.2 MATLAB インターフェースの応用

MIP を使えば、ASCET から MATLAB の全機能を利用できます。利用できる機能には、基本パッケージの基本関数、MATLAB ツールボックスのすべての関数、および MATLAB M- ファイルの形式で作成されたすべてのユーザ定義関数などがあります。Simulink モデルのシミュレーションを ASCET から開始することもできるので、ASCET と Simulink の結合（コシミュレーション）が可能です（ただし、オンラインのリアルタイム操作においては不可能です）。



以下に、MATLAB の応用例について概説します。

MATLAB ー ASCET 間でデータ交換を行う

ASCET データは MATLAB に送信され、シミュレーションが中断されている間は、MATLAB ワークスペース 内でデータの加工、演算の実行、データの視覚化などの処理をマニュアル操作で行うことができます。必要に応じて ASCET はその計算結果を MATLAB から受信し、シミュレーションの続行時にそれを使用することができます。

MATLAB 関数をASCET モデルに組み込む

ASCET でのシミュレーション実行中は、データは MATLAB に周期的に送信され、リモート操作でリクエストされる MATLAB 関数によって自動的に処理されます。データは、シミュレーションステップごとに 1 回ずつ MATLAB に転送したり、あるいは ASCET でバッファリングして、N ステップ分をデータパケットとしてまとめて転送することもできます。シミュレーションは MATLAB からの処理結果が届くまでは待ち状態となり、届いた結果に基づいて、たとえば ASCET のモデルパラメータの調整、といった処理を行い、その後、シミュレーションが続行されます。

Simulink モデルをASCET モデルに組み込む

システムの一部、たとえばプラントモデルを MATLAB/Simulink で記述し、コントローラ部分を ASCET に実装します。Simulink モデルへの入力（つまりここではコントローラ出力）は MATLAB に送信され、そのモデル用の積分ステップが Simulink 内で実行されます。MATLAB から受け取った Simulink モデルからの出力（プラントモデルからの出力）は ASCET モデル（コントローラ）に渡され、そこで次の積分ステップ用のコントロール値が算出されます。

4 MATLAB インターフェース ー ASCET からの呼び出し

MATLAB インターフェースの機能は、モデリングレベルにおいても実験レベルにおいても利用できます。

モデリングレベルの場合、MATLAB へのリンクは ASCET 内の C コード、あるいは連続系モデルの場合は C コードブロックの形で実現されます。これにより、ASCET モデル内のどこでも MATLAB を使用でき、MATLAB 関数を API の形で ASCET に直接組み込むことも可能です。

実験環境では、ASCET シミュレーションステップの前後で、MATLAB インターフェース関数を周期的に実行することができます。これを行うためのウィンドウが提供されていて、その中に送受信する ASCET 変数のリストを登録することができます。また別のリストに、MATLAB 内で実行される MATLAB 関数を登録することができます。ただし、モデリングレベルでの C コードブロックの場合に比べて応用範囲は限定されます。

4.1 モデリングレベル

一般に、MATLAB インターフェース関数の利用方法は、C コードブロック内で任意に使用される C 関数を利用する場合と同じです。MIP は C ライブラリ（MATLAB へのインターフェースを提供する DLL として実装されます）で、標準 C 言語に関数を追加して機能を拡張する形をとっています。

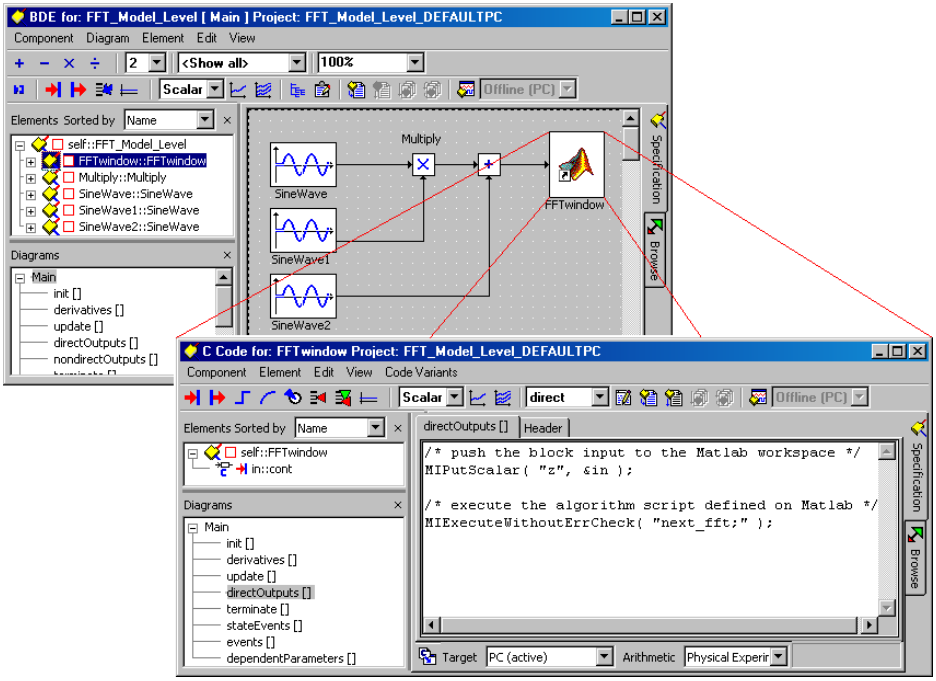
4.1.1 ASCET 内での C コードブロック

ブロックの中には 2 つのメソッドを定義する必要があります。まず、初回にのみ実行されるメソッドから MIOpen を呼び出し、MATLAB へのリンクを確立します。ここでは必要に応じて、さらに MATLAB ワークスペース や ASCET に値のプリセットを行うための関数を実行することもできます。そしてもう 1 つのメソッドに、周期的な各種リクエスト（MATLAB へのデータ送信、MATLAB 関数の実行、データ受信）を実行するコードを組み込みます。各メソッドの実行シーケンスは、モデル全体として、シーケンスコールの形で適切に定義する必要があります。

1 つの ASCET モデルの中に MIP 関数を使用する C コードブロックが複数ある場合でも、`MIOpen` を呼び出すのは 1 回だけです。必ず、最初に実行される MIP C コードブロックが `MIOpen` コールを実行するようにしてください。また、複数の C コードブロックで同名の MATLAB 変数を使用する場合には注意が必要です。すべてのブロックが同じ MATLAB プロセスを使用して通信するため、使用される変数はそれをコールする C コードブロック単体にリンクされるのではなく、モデル全体に公開されてしまいます。

4.1.2 C コードの CT ブロック

C コードで記述された MIP の CT ブロックは、固定的なメソッド群で構成されます。ASCET の C コードブロックの場合と同様に、`init[]` メソッドで `MIOpen` リクエストを実行します。周期的に実行されるコードは、`directOutputs[]` や `update[]` などのメソッド内に含めます。入出力は、適宜に定義します。



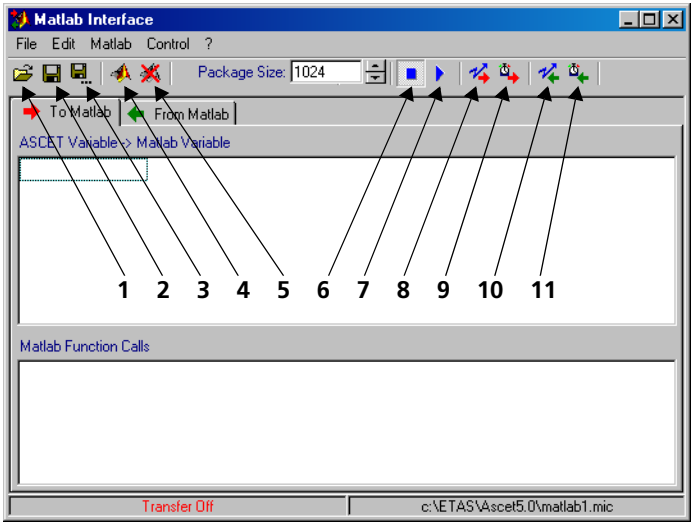
ASCET の C コードのクラスやモジュールの場合と同様、複数の C コードの CT ブロックを使用する場合も `MIOpen()` のリクエストは一度だけでよく、MATLAB ワークスペース の変数はグローバルなアクセスが可能です。

4.2 実験レベル

C コードブロックに加え、ASCET にはもうひとつの MATLAB インターフェースがあります。ASCET の実験環境の **Matlab** メニューによって、“Matlab Interface” ダイアログウィンドウが開きます。このウィンドウから、MATLAB の起動や、コマンドやデータの転送を行うことができます。

4.2.1 データ転送と MATLAB ファンクションコール用のウィンドウ

ASCET の実験環境のメニューバーから **Matlab → Open Interface** を選択すると、ツールボックスとしての機能を備えた“Matlab Interface”ウィンドウが開きます。



現バージョンでは、以下のボタンを使用できます（ボタンで実行されるアクションは、ボタン名の右に併記されているメニューコマンドによっても実行することもできます）。

1. **Load Settings**（メニューコマンド：**File → Load Settings**）
   ファイルに保存されている MATLAB インターフェースウィンドウの設定（*ASCET ←→ MATLAB の変数割り当て、MATLAB ファンクションコール*）をロードします。
2. **Save Settings**（メニューコマンド：**File → Save Settings**）
   MATLAB インターフェースウィンドウの現在の設定をファイルに保存します。
3. **Save Settings As**（メニューコマンド：**File → Save Settings As**）
   MATLAB インターフェースウィンドウの現在の設定を、指定した名前のデータ形式で格納します。
4. **Open Matlab**（メニューコマンド：**Matlab → Open Matlab**）
   MATLAB を起動して、通信を確立します。
5. **Close Matlab**（メニューコマンド：**Matlab → Close Matlab**）
   MATLAB との通信を終了し、MATLAB をシャットダウンします。

6. **Stop Cyclic Transfer**（メニューコマンド：**Control → Stop Transfer**）
   ASCET から MATLAB へ、あるいは MATLAB から ASCET への周期的なデータ転送を停止します。
7. **Start Cyclic Transfer**（メニューコマンド：**Control → Start Transfer**）
   周期的なデータ転送を開始します。これを行うには、シミュレーションがあらかじめ実行されている必要があります。変数割り当てリストに登録されている変数の値が ASCET 内で集められ、指定のシミュレーションステップ数が経過した後に、ASCET から MATLAB へ、あるいは MATLAB から ASCET へ転送されます。データ転送ブロックのサイズは、MATLAB インターフェースウィンドウの“Package Size”という入力フィールドで指定します。

   “**MATLAB Function Calls”フィールドに指定された** MATLAB アクションは、MATLAB へのデータ転送終了後に実行されます。

注記

上記の内容は、`ToMatlab` メッセージと `FromMatlab` メッセージの両方に当てはまります。

以下のボタンは、マニュアル操作による MATLAB とのデータ送受信に使用します。ウィンドウのメニューには、これらのボタンに相当するメニューコマンドはありません。

8. **Init To Matlab**
   “ToMatlab”タブ内の `Init` 属性を持つ変数アクションあるいは MATLAB ファンクションコールを実行します。
9. **Send To Matlab**
   ASCET から MATLAB への転送を実行します。これは、“To Matlab”タブに入力されたすべての ASCET → MATLAB 変数割り当てが対象となります。変数の転送が終わると、“Matlab function calls”フィールドに入力されている MATLAB コマンドが順に実行されます。
10. **Init From Matlab**
    “From Matlab”タブ内の Init 属性を持つ変数のアクションあるいは MATLAB ファンクションコールが実行されます。
11. **Receive From Matlab**
    MATLAB から ASCET への転送を実行します。これは、“From Matlab”タブに入力されたすべての ASCET ← MATLAB 変数割り当てが対象となります。“Matlab function calls”フィールドに入力されている MATLAB コマンドが順に実行された後、変数が MATLAB から ASCET へ転送されます。

ボタンで操作されるすべてのコマンドやアクションは、ウィンドウのメニューによっても操作することができます。

ASCET 変数を *“To Matlab”* と *“From Matlab”* *の 2 つのタブの変数フィールド*に割り当てるには、ドラッグ＆ドロップ機能を利用ことができます。デフォルトでは、ASCET の変数名はそのまま MATLAB の変数名として使用することができます

が、デフォルトの MATLAB 変数名がバックスラッシュ（“¥”）を含んでいる場合は ASCET では使用できないので、変数名をダブルクリックしてマニュアル操作で修正してください。

MATLAB ファンクションコールの記述を行うには、メニューコマンド **Edit → Function Calls → Add Function Call**、あるいは“Matlab Function Call”フィールドをマウスで右クリックして表示されるショートカットメニューの **Add Function Call** を選択します。*init*、*cyclic*、*disabled* というプロパティを関数と変数に割り当てます。init プロパティが割り当てられた変数とファンクションコールは、MATLAB インターフェースが開いた直後に 1 回だけ、あるいは **Init To Matlab** (8) や **Init From Matlab** (10) ボタンが押された時に、転送あるいは実行されます。

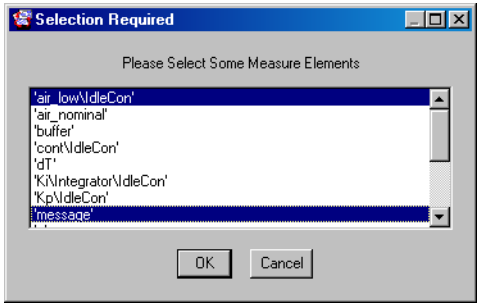
MATLAB とのデータ交換はブロック単位で行われます。つまり、タスクが実行された後に各出力信号はバッファに書き込まれ、バッファがフルになった時点で MATLAB への転送が開始されます。この Package Size は任意に変更できます。

データを転送する際は、すでに ASCET の実験ウィンドウからシミュレーションが開始されている必要があります。ボタン 4 および 7 によって、MATLAB に転送するデータの収集が開始されます。Package Size に指定された量のデータが蓄積されると直ちにパッケージが MATLAB に送信され、次に、*Function Calls* リストが実行されます。このようにして、ASCET から MATLAB への転送データ（“To Matlab”タブ）の送信、MATLAB でのアクション実行、そして MATLAB から ASCET への転送データ（“From Matlab”タブ）の受信、という順で処理が行われます。

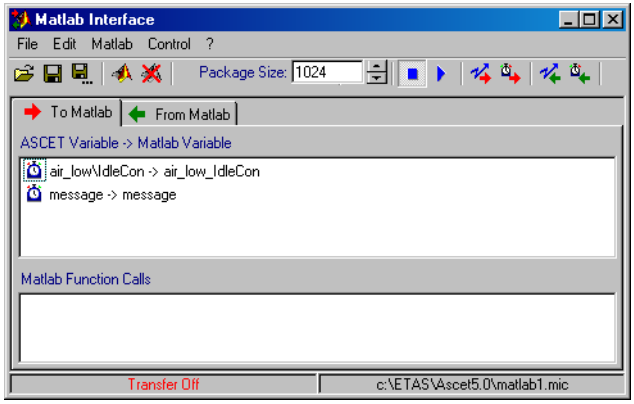
4.2.2 MATLAB とのデータ送受信

変数： MATLAB に送信される変数や、MATLAB プログラムから受信する変数の名前は、“To Matlab”タブの“ASCET Variable → Matlab Variable”フィールド、または“From Matlab”タブの“ASCET Variable ← Matlab Variable”フィールドに入力します。

変数は、ASCET 実験環境から *ドラッグ& ドロップで “To MATLAB” タブ*に追加できます。またあるいは、“Matlab Interface”ダイアログの変数フィールドでマウスの右ボタンをクリックして*表示される*ショートカットメニューで **Add Element** を選択すると、ASCET 変数が選択ボックスに表示されます。

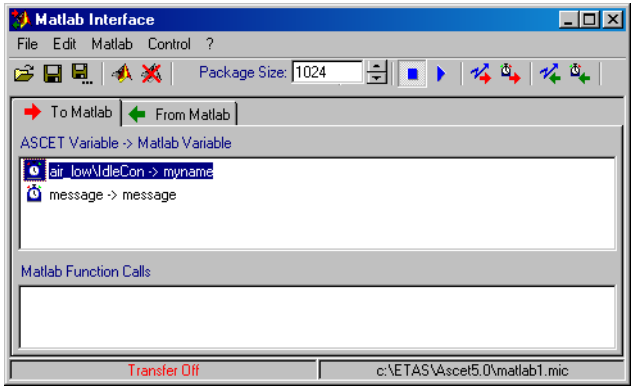


ここでは複数の変数を選択することが可能です。必要な変数をすべて選択し、**OK** ボタンで確定します。



選択された ASCET 変数が、“To Matlab” タブの “ASCET Variable → Matlab Variable” フィールドに表示されます。割り当てられた MATLAB 変数は、デフォルトでは ASCET 変数と同じ名前になります。

MATLAB でのデフォルト名は、‘¥' や ‘/' が ‘_' に置き変わる以外は、ASCET での名前と同じです。MATLAB 変数名を変更するには、変更したい変数をダブルクリックしてから、新しい名前を入力します。



この例では、ASCET 変数の `air_low` が MATLAB 変数 `myname` に割り当てられています。`air_low¥test_class_C_1` という名前は、変数 `asd_a` の内部名です。

各変数には、メニューコマンド **Edit → Element**、またはショートカットメニューで、**Init**、**Cyclic**、**Disabled** のいずれかの属性を割り当てることができます。この属性によって、その変数を転送するかしないか、あるいは最初に一度だけ転送するかどうかが決まります。

- **Init**：

変数の値は、シミュレーション開始時に一度だけ、自動的に MATLAB に転送されます。

- **Cyclic**：

変数の値が転送されます。

- **Disabled**：

変数の値は転送されません。

“From Matlab”タブの“ASCET Variable ← Matlab Variable”フィールドに MATLAB から受信されるデータアイテムを割り当てる際も、上記の方法と同じ方法で行います。

4.2.3 MATLAB 関数の実行

“To Matlab”および“From Matlab”タブ上の MATLAB コマンドリストには、それぞれもう 1 つのリストが付随しています。そのリストの中には、MATLAB へ変数を送信した後、あるいは MATLAB から変数を受信する前に実行される MATLAB 関数を、その転送パラメータとともに登録します。登録された関数は変数に直接割り当てられるのではなく、データ送信と受信との間に関数リストとして実行されます。ここでもやはり、ファンクションコールに属性を割り当てて、コールの実行について明確に定義することができます（22 ページの「変数」を参照してください。）

- **Cyclic** / **Disabled**：

実行を許可／不許可にします。

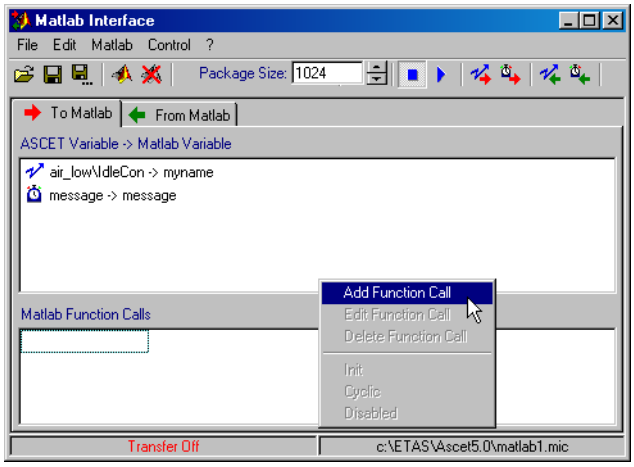
- **Init**：

MATLAB との接続確立時に一度だけ実行されます。

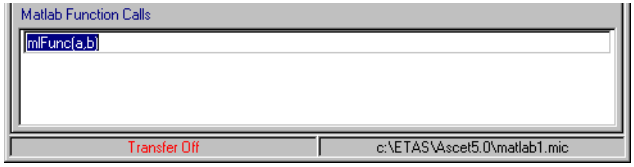
MATLAB ファンクションコールの追加は、以下のように行います。

MATLAB ファンクションコールを追加する：

- “Matlab Function Calls”リストエリアを右クリックして、ショートカットメニューから **Add Function Call** を選択します。

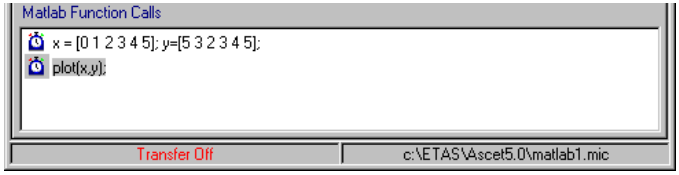


選択が完了すると、ファンクションコールの標準エントリが作成されてリスト内に表示されます。下図は、*デフォルトのファンクションコール*が新たに追加された状態を示しています。



- 新しく作成されたファンクションコールのデフォルト名を変更するには、そのファンクションコールをダブルクリックしてから新しい名前を入力します。

ここでは、有効な MATLAB コールやコマンドのすべてを使用できます。以下の例では、2 つの *MATLAB ファンクションコール*が使用されています。



*MATLAB ファンクションコール*は、“To Matlab”タブおよび“From Matlab”タブ上で、互いに独立して指定することができます。Start ボタン（7）で周期的なデータ転送を開始すると、“To Matlab”タブに指定された変数が転送され、“To Matlab”タブの MATLAB ファンクションコールが実行されます。続いて“From Matlab”タブの MATLAB ファンクションコールが実行され、最後に“From Matlab”タブの変数が転送されます。

すべての MATLAB 関数（`*.m` ファイル）をファンクションコールとして使用することができます。その場合、変数名の対応と MATLAB スクリプトの妥当性は、ユーザーの責任において管理する必要があります。

4.2.4 重複時の対処

リスト内の複数の変数に同じ属性が与えられている場合、リスト内で上位にある変数／関数が先に処理されます。

注記

ウィンドウ内の Start と Stop ボタンは、ASCET 実験のシミュレーションの実行には影響せず、データ転送制御にだけ影響します。
ASCET シミュレーションの制御は、“Physical Experiment”ウィンドウのボタンあるいはメニューから行います。

5 MATLAB インターフェース ― リファレンス

5.1 プロセス間通信

本項で紹介する関数を使って、MATLAB プロセスのリモート起動や、MATLAB への通信リンクの開始／終了が行えます。

`MIExecute`、`MIExecuteAndEcho`、および `MIExecuteWithoutErrCheck` は、MATLAB 関数をリモートで呼び出して実行するための関数です。これらの関数を使って MATLAB の全機能を利用することができます。

`MIExecute` および `MIExecuteAndEcho` は MATLAB の内部エラーのチェックとエラー処理を行います。このエラー処理では、`MIExecuteWithoutErrCheck` が重要な役割を果たします（詳細は、5.6 章「エラー処理」を参照してください。）。

`MIExecuteWithoutErrCheck` を**使用**することは避けてください。。

エラー処理の手続き上、MATLAB エンジンによって提供される `MIExecute`、および `MIP` 関数の `MIActivateBuffer`（5.7 章「内部関数」および『MATLAB API Guide』を参照してください）を実行した際、文字バッファ内に格納された MATLAB 出力を応答として読み取ることは、事実上行えません。代わりに `MIExecuteAndEcho` を使用してください。

5.1.1 `MIOpen`

構文：

```
bool MIOpen( void )
```

引数：

なし。

成功すれば `true` を、そうでなければ `false` を返します。

機能：

MATLAB プロセスを開始し、MATLAB への通信リンクを確立します。

例：

```
ok = MIOpen();
```

参照：

→ `"MIClose"`

5.1.2 MIClose

構文：

```
bool MIClose( void )
```

引数：

なし。

成功すれば true を、そうでなければ false を返します。

機能：

実行していた MATLAB プロセスを終了し、接続を切断します。

例：

```
ok = MIClose();
```

参照：

→ "MIOpen"

5.1.3 MIExecute

構文：

```
bool MIExecute( char *evalstring )
```

引数：

evalstring	文字列

機能：

evalstring は、MATLAB で実行される MATLAB コードが格納された文字列で、この関数には MATLAB の任意の式または文を記述できます。MIExecute は MATLAB 環境内でエラーが発生したかどうかを調べ、エラーが検出されるとエラーメッセージを発行します。

成功すれば true を返し、そうでなければ false を返します。

例：

```
/* MATLAB の図中に正弦波を描画します。 */
MIExecute("t = 0:0.01:5");
MIExecute("f = 2.7818");
MIExecute("y = sin(2*pi*f*t)");
MIExecute("plot(t, y, 'r')");
/* my_MATLAB_function.m という M- ファイルに定義されている */
```

```
/* ユーザ定義の MATLAB 関数を実行します。 */
MIExecute("out = my_Matlab_function(arg1, arg2, argN)");
```

参照：

→ "MIExecuteAndEcho"

5.1.4 MIExecuteAndEcho

構文：

```
bool MIExecuteAndEcho(  char *evalstring,
                        char *buffer )
```

引数：

evalstring	文字列
buffer	文字列

機能：

MIExecute() と同様ですが、バッファ buffer に MATLAB 出力をエコーします。

注意：

buffer として宣言されているサイズよりも MATLAB 出力の実際のサイズの方が大きいと、システムクラッシュが発生します。

成功すれば true を、そうでなければ false を返します。

例：

```
char outputBuffer[1024];

MIExecuteAndEcho("A = [1 2;-3 -4 ]; eig(A)",
outputBuffer);
printf("Eigen values of A: ¥n %s¥n",
outputBuffer);
```

参照：

→ "MIExecute"

5.1.5 MIExecuteWithoutErrCheck

構文：

```
bool MIExecuteWithoutErrCheck( char *evalstring )
```

引数：

evalstring	文字列

機能：

この関数の機能は MIExecute と同様ですが、MATLAB 環境で発生するエラーはチェックされません。MIExecuteWithoutErrCheck は、MATLAB コードにまったくエラーがないことが確認できていて、さらに実行時エラーが起こってもそれほど重大な事態にはならない場合に限り使用するようにしてください。エラーチェックを行う場合よりも短い時間で実行できるので、実行時間を短縮できる場合があります。

例：

```
/* 以下のコマンドを実行して、2 つの関数の相違を比べてみます。*/

/*  A は 3 × 2 のマトリックスですが、*/
/*  行列式（determinant）は正方行列しか */
/*  サポートしていないので、b を算出しようとすると */
/*  MATLAB エラーになります。*/

/* エラーチェックは行われないので、 */
/* エラーメッセージは表示されません。*/
MIExecuteWithoutErrCheck
   ("A = [1 2 3; 4 5 6]; b = det(A);");

/* エラーメッセージウィンドウが表示され、*/
/* MATLAB エラーメッセージが表示されます。*/
MIExecute("A = [1 2 3; 4 5 6]; b = det(A);");
```

参照：

→ "MIExecute"

→ "MIExecuteAndEcho"

注記

MATLAB コードのエラーなどによってランタイムエラーが発生すると、その次に実行された以下のいずれかの関数のエラーとして通知されます。

```
MIExecute(),        MIExecuteAndEcho(),
MIGetComplexMatrix(),MIGetComplexResult(),
MIGetComplexScalar(),MIGetComplexVector(),
MIGetMatrix(),      MIGetNCols(),
MIGetNRows(),       MIGetResult(),
MIGetScalar(),      MIGetString,
MIGetVector()
```

5.2 データ管理

ASCET で扱えるデータ型は、実数のスカラ、ベクトル（array）、およびマトリックスのみですが、MATLAB では他にいくつかのデータ型を利用できます。

送受信の処理ができるのは、C 言語で記述された倍精度型のスカラと配列（ベクトルとマトリックスを含みます）だけです。

MATLAB で算出された複素数を扱うためには、データの実数部分と虚数部分を別の変数として送受信する必要があります。MATLAB ワークスペース 内では、複素数データは 1 つの構造体として表現されています。

MI 関数に渡される引数は倍精度のスカラまたは配列を参照するポインタです。そのため、使用する変数用のメモリを確保したり解放したりする処理が必要です。これは明示的な C コードの外部に ASCET クラス変数を定義するか、あるいはその変数のスコープが C コードのクラス／ CT ブロックに限られている場合には後述の関数を呼び出すことによって行うことができます。

注記

MI 関数名や本書で、「ベクトル」という語は、ASCET の配列（array）型、およびそれに対応する MATLAB 構造体に対して使用されています。

5.2.1 MICreateDouble

構文：

```
MICreateDouble( aVarName, nElements )
```

引数：

aVarName	変数の名前
nElements	確保するエレメントの数

機能：

MICreateDouble は、すでに宣言されている double 型変数 aVarName の nElements 個分のメモリを確保するマクロです。

確保したメモリは、MIDestructDouble を実行して解放してください。

例：

```
double *aVariable;

MICreateDouble(aVariable, 9);

MIExecute("A=magic(3);");
MIGetMatrix(aVariable, "A");

MIDestructDouble(aVariable);
```

参照：

→ "MIDestructDouble"

5.2.2 MIDestructDouble

構文：

```
MIDestructDouble( aVarName )
```

引数：

aVarName	変数の名前

機能：

MIDestructDouble は、参照される変数 aVarName に割り当てられているメモリを解放します。メモリは、MICreateDouble を実行することによって割り当てられています。

例：

```
MICreateDouble(aVariable,9);
...
```

```
MIDestructDouble(aVariable);
```

参照：

→ "MICreateDouble"

5.3 MATLAB へのデータ送信

本項で紹介する関数は、MATLAB ワークスペース へのデータ送信（***put***）処理をサポートします。実数データと複素数データの転送は、それぞれ別の関数で行われます。送信できるのは、マトリックス、ベクトル、およびスカラのデータに限られ、それらに対応する MATLAB データ構造体が MATLAB 環境内に作成されます。使用できるデータ型についての詳細は、データ管理の章を参照してください。

5.3.1 MIPutMatrix

構文：

```
bool MIPutMatrix(  char *MatlabVarName,
                   double *rVar,
                   int nRows,
                   int nCols )
```

引数：

MatlabVarName	データが代入される MATLAB 変数の名前
rVar	転送される実数へのポインタ
nRows	データが代入される MATLAB マトリックスの行数
nCols	データが代入される MATLAB マトリックスの列数

機能：

MATLAB にマトリックスデータを送信し、MatlabVarName で指定された変数に代入します。rVar により参照されるデータは double 型でなければなりません。nRows と nCols は、MATLAB ワークスペース 内に作成されるベクトルのサイズを示します。

成功すれば true を、そうでなければ false を返します。

例：

```
/* データの定義 */
double rdata[] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34};

/* rdata を送信し、 */
/* MATLAB 変数 Fibo に代入します。 */
```

```
MIPutMatrix("Fibo", rdata, 2, 5);
```

Matlab ワークスペース 内では：

```
% Fibo は 5x2 のマトリックスです。
>> Fibo =
     0   1   3   8   21
     1   2   5  13   34
```

参照：

→ "MIPutVector"

→ "MIPutScalar"

→ "MIPutComplexMatrix"

→ "MIPutComplexVector"

→ "MIPutComplexScalar"

5.3.2 MIPutScalar

構文：

```
bool MIPutScalar(  char *MatlabVarName,
                   double *rVar )
```

引数：

MatlabVarName	データが代入される MATLAB 変数の名前
rVar	転送される実数データへのポインタ

機能：

MATLAB にスカラデータを送信し、MatlabVarName で指定された変数に代入します。rVar により参照されるデータは double 型でなければなりません。内部的には、この関数は受け取った引数を nRows = 1 および nCols = 1 という固定パラメータとともに、関数 MIPutMatrix に渡します。

成功すれば true を、そうでなければ false を返します。

例：

```
/* データの定義 */
double realScalar = 5;

/* realScalar を送信し、*/
/* MATLAB 変数 x に代入します。*/
```

```
MIPutScalar("x", &realScalar);
```

参照：

→ "MIPutMatrix"

→ "MIPutVector"

→ "MIPutComplexMatrix"

→ "MIPutComplexVector"

→ "MIPutComplexScalar"

5.3.3 MIPutVector

構文：

```
bool MIPutVector(  char *MatlabVarName,
                   double *rVar,
                   int nRows )
```

引数：

MatlabVarName	データが代入される MATLAB 変数の名前
rVar	転送される実数データへのポインタ
nRows	データが代入される MATLAB ベクトルの行数

機能：

MATLAB に配列データを送信し、MatlabVarName で指定された変数に代入します。rVar により参照されるデータは double 型でなければなりません。nRows は、MATLAB ワークスペース 内に作成されるベクトルの要素数を示します。この関数は受け取った引数を nCols = 1 という固定パラメータとともに、関数 MIPutMatrix に渡します。

成功すれば true を、そうでなければ false を返します。

例：

```
/* データの定義 */
double rdata[] = {0, 1, 1, 2, 3, 5, 8, 13, 21,34};


/* rdata を送信し、*/
/* MATLAB 変数 Fib に代入します。*/
MIPutVector("Fib", rdata, 10);
```

Matlab ワークスペース 内では：

```
% Fib はエレメント数が 10 の列ベクトルです。
Fib =
0
1
1
2
3
5
8
13
21
34
```

参照：

→ "MIPutMatrix"
→ "MIPutScalar"
→ "MIPutComplexMatrix"
→ "MIPutComplexVector"
→ "MIPutComplexScalar"

5.3.4 MIPutComplexMatrix

構文：

```
bool MIPutComplexMatrix(char *MatlabVarName,
                        double *varReal,
                        double *varImag,
                        int nRows,
                        int nCols )
```

引数：

MatlabVarName	データが代入される MATLAB 変数の名前
`varReal`	転送されるデータの実数部分へのポインタ
`varImag`	転送されるデータの虚数部分へのポインタ
`nRows`	データが代入される MATLAB マトリックスの行数
`nCols`	データが代入される MATLAB マトリックスの列数

機能：

MATLAB に 2 つの配列データを送信し、`MatlabVarName` で指定された変数の実数部分と虚数部分に代入します。`nRows` と `nCols` は、MATLAB ワークスペース内に作成されるベクトルのサイズを示します。

成功すれば `true` を、そうでなければ `false` を返します。

例：

```
double outReal[] = {2, 3, 5, 7};
double outImag[] = {11, 13, 17, 19};

/* varReal および varImag のデータを送信し、*/
/* MATLAB 変数 C に代入します。*/
MIPutComplexMatrix("C", outReal, outImag, 2, 2);
```

Matlab ワークスペース 内では：

```
>> C =
          2 + 11i    5 + 17i
          3 + 13i    7 + 19i
```

参照：

→ "MIGetComplexResult"
→ "MIPutComplexVector"
→ "MIPutComplexScalar"
→ "MIGetComplexMatrix"
→ "MIGetComplexVector"
→ "MIGetComplexScalar"

5.3.5 MIPutComplexScalar

構文：

```
bool MIPutComplexScalar(char *MatlabVarName,
                        double *varReal,
                        double *varImag )
```

引数：

MatlabVarName	データが代入される MATLAB 変数の名前
varReal	転送されるデータの実数部分のポインタ
varImag	転送されるデータの虚数部分のポインタ

機能：

2 つのスカラ（1 つが実数部分を表し、もう 1 つが虚数部分を表す）で表現される複素数のスカラデータを MATLAB に送信し、MatlabVarName で指定された変数の実数部分と虚数部分に代入します。内部的には、この関数は渡された引数を nRows = 1 および nCols = 1 という固定的なパラメータとともに、関数 MIPutComplexMatrix に渡します。

成功すれば true を、そうでなければ false を返します。

例：

```
double scalarReal = 1;
double scalarImag = -4;

/* scalarReal および scalarImag のデータを送信し、*/
/* MATLAB 変数 E に代入します。*/
MIPutComplexScalar("E", &scalarReal,
   &scalarImag);
```

参照：

→ "MIGetComplexResult"
→ "MIPutComplexMatrix"
→ "MIPutComplexVector"
→ "MIGetComplexMatrix"
→ "MIGetComplexVector"
→ "MIGetComplexScalar"

5.3.6 MIPutComplexVector

構文：

```
bool MIPutComplexVector(char *MatlabVarName,
                        double *varReal,
                        double *varImag,
                        int nRows )
```

引数：

MatlabVarName	データが代入される MATLAB 変数の名前
varReal	転送されるデータの実数部分へのポインタ
varImag	転送されるデータの虚数部分へのポインタ
nRows	データが代入される MATLAB マトリックスの行数

機能：

2 つの配列で表現される複素数のベクトルデータを MATLAB に送信し、MatlabVarName で指定された変数の実数部分と虚数部分に代入します。nRows は、MATLAB ワークスペース 内に作成される配列のサイズを示します。内部的には、この関数は渡された引数を nCols = 1 という固定パラメータとともに、関数 MIPutComplexMatrix に渡します。

成功すれば true を、そうでなければ false を返します。

例：

```
double outReal[] = {2, 3, 5, 7};
double outImag[] = {11, 13, 17, 19};

/* outReal および outImag のデータを送信し、*/
/* MATLAB 変数 C に代入します。*/
MIPutComplexVector("C",  outReal, outImag, 4);
```

参照：

→ "MIGetComplexResult"

→ "MIPutComplexMatrix"

→ "MIPutComplexScalar"

5.4 MATLAB からのデータ受信

本項では、MATLAB ワークスペース からデータを受信（***get***）する方法について説明します。2 種類のデータ転送方法があり、渡される引数の型は、以下に述べるようにそれぞれ異なります。引数の型の整合性がとれていないと、システムが予期しない動作をしたり、プロセスが終了（システムクラッシュ）してしまう可能性があります。

MIGetResult、*MIGetComplexResult*

MATLAB は、転送されたデータを MATLAB 固有の C 構造体として格納するためのメモリ領域を共有メモリ内に自分自身で確保するので、`MIGetResult` および `MIGetComplexResult` 関数実行時には、データを格納するためのメモリを確保する必要がありません。データを転送するために 2 つのメソッドが提供されています。2 つのメソッドは引数が異なり、引数が正しく指定されていないと、システムが予期しない動作をしたり、プロセスが終了（システムクラッシュ）してしまう可能性があります。

MIGetMatrix、*MIGetVector*、*MIGetScalar*、*MIGetComplexMatrix など*

これらの関数では、関数の引数用のメモリに MATLAB 構造体のデータが明示的にコピーされるので、あらかじめメモリを確保しておく必要があります。

内部的には主に `MIGetString` が使用されますが、これは無制限に使用できます。

5.4.1 `MIGetResult`

構文：

```
bool MIGetResult(  double **result,
                   char *MatlabVarName )
```

引数：

`result`	double 型へのポインタを参照するポインタ
`MatlabVarName`	文字列

機能：

MATLAB ワークスペース から ASCET 環境へデータを転送します。

`result` は、受信データを参照する `double` のポインタへのポインタです。すでに MATLAB がデータ用のメモリを確保しているので、この関数を使用する前に `MICreateDouble` を実行する必要はありません。

`MatlabVarName` は転送されるデータの MATLAB 変数名が格納されている文字列です。MATLAB 変数は 2 次元マトリックス、ベクトル、あるいはスカラでなければなりません。`result` の型、ディメンション、サイズを調べるには、`"MIIsReal"`、`"MIGetNRows"`、`"MIGetNCols"` を使用してください。

MIGetResult は参照される MATLAB 変数の実数部分だけを転送します。複素数の転送については、"MIGetComplexResult" を参照してください。

成功すれば true を、そうでなければ false を返します。

例：

```
double *in;

MIExecute("A = [1 2; -3 -4]; e = eig(A);");
MIGetResult(&in, "e");
```

参照：

→ "MIGetComplexResult"

→ "MIGetMatrix"

→ "MIGetVector"

→ "MIGetScalar"

→ "MIGetString"

→ "MIGetNRows"

→ "MIGetNCols"

→ "MIIsReal"

→ "MIIsComplex"

→ "MIIsEmpty"

5.4.2 MIGetMatrix

構文：

```
bool MIGetMatrix( double *in, char *MatlabVarName )
```

引数：

in	double 型へのポインタ
MatlabVarName	文字列

機能：

MATLAB ワークスペース から ASCET 環境へ、マトリックスを転送します。

in は、double 型の受信データへのポインタです。参照する ASCET マトリックスがまだ宣言されていない場合、この関数を実行するためには、MICreateDouble を実行して受信データ用に十分なメモリを確保しておく必要があります。

MatlabVarName は、転送されるデータの MATLAB 変数名が格納されている文字列です。この MATLAB 変数は適切なサイズの 2 次元マトリックスでなければなりません。in の型、ディメンション、サイズを調べるには、"MIIsReal"、"MIGetNRows"、"MIGetNCols" を使用してください。

MIGetMatrix は参照される MATLAB 変数の実数部分だけを転送します。複素数型のマトリックスを転送する場合は、"MIGetComplexMatrix" を参照してください。

成功すれば true を、そうでなければ false を返します。

例：

例 1：

```
/* asdMatrix は、C コードクラスの */
/* エレメントリスト内で宣言されている */
/* ユーザー定義の ASCET 変数です。*/

MIExecute("A = magic(3);");
MIGetMatrix(asdMatrix, "A");
```

例 2：

```
/* in はユーザー定義のローカル変数で、*/
/* ASCET で見ることはできません。*/
/* そこで、そのためのメモリを */
/* 確保する必要があります !*/

double *in;
MICreateDouble(in, 9);
MIGetMatrix(in, "A");
MIDestructDouble(in);
```

参照：

→ "MIGetComplexMatrix"

→ "MIGetVector"

→ "MIGetScalar"

5.4.3 MIGetVector

構文：

```
bool MIGetVector(  double *in,
```

```
                    char *MatlabVarName )
```

引数：

`in`	double 型へのポインタ
`MatlabVarName`	文字列

機能：

MATLAB ワークスペース からユーザー環境へ、ベクトルを送信します。

`in` は、`double` 型への受信データへのポインタです。参照する ASCET 配列がまだ宣言されていない場合、この関数を実行するためには、`( MICreateDouble` を実行して受信データ用に十分なメモリを確保しておく必要があります。

`MatlabVarName` は、転送されるデータの MATLAB 変数名が格納されている文字列です。この MATLAB 変数は適切なサイズのベクトルでなければなりません。`in` の型とサイズを調べるには、`"MIIsReal"`、`"MIGetNRows"`、`"MIGetNCols"` を使用してください。

`MIGetVector` は参照される MATLAB 変数の実数部分だけを転送します。複素数型のベクトルを転送する場合は、`"MIGetComplexVector"` を参照してください。

成功すれば `true` を、そうでなければ `false` を返します。

例：

例 1：

```
/* asdArray は、C コードクラスエレメントリスト内で */
/* 宣言されているユーザー定義の */
/* ASCET 変数です。*/

MIExecute("randomVec = rand(9,1);");
MIGetVector(asdArray, "randomVec");
```

例 2：

```
/* in はユーザー定義のローカル変数で、*/
/* ASCET で見ることはできません。*/
/* そこで、ユーザーはそのためのメモリを */
/* 確保する必要があります！  */

double *in;
MICreateDouble(in, 9);
```

```
MIExecute("randomVec = rand(9,1);");
MIGetVector(in, "randomVec");
MIDestructDouble(in);
```

参照：

→ "MIGetComplexVector"

→ "MIGetMatrix"

→ "MIGetScalar"

5.4.4 MIGetScalar

構文：

```
bool MIGetScalar(  double *in,
                   char *MatlabVarName )
```

引数：

in	double 型へのポインタ
MatlabVarName	文字列

機能：

MATLAB ワークスペース から ASCET 環境へ、スカラデータを送信します。

in は、double 型の受信データを参照するためのポインタです。

MatlabVarName は、転送されるデータの MATLAB 変数名が格納されている文字列です。この MATLAB 変数はスカラでなければなりません。

MIGetScalar は参照される MATLAB 変数の実数部分だけを転送します。複素数型のスカラを転送する場合は、"MIGetComplexScalar" を参照してください。

成功すれば true を、そうでなければ false を返します。

例：

例 1：

```
/* asdCont は、C コードクラスエレメントリスト内で */
/* 宣言されているユーザー定義の */
/* ASCET 変数です */

MIExecute("MATLABVar = pi;");
MIGetScalar(&asdCont, "MablabVar");
```

例 2：

```
/* in はユーザー定義のローカル変数で、*/
/* ASCET で見ることはできません。*/
/* しかしスカラの場合は、明示的にメモリを */
/* 確保する必要はありません！ */

double *inVar;

MIExecute("MATLABVar = pi;");
MIGetScalar(&inVar, "MATLABVar");
```

参照：

→ "MIGetComplexScalar"

→ "MIGetMatrix"

→ "MIGetScalar"

5.4.5 MIGetComplexResult

構文：

```
bool MIGetComplexResult(double **inReal,
                        double **inImag,
                        char *MatlabVarName )
```

引数：

inReal	double 型へのポインタを参照するポインタ
inImag	double のポインタへのポインタ
MatlabVarName	文字列

機能：

MATLAB ワークスペース からユーザー環境へ複素数データを転送します。

inReal および inImag は、すでに内部的にメモリ確保されている受信データの実数部分と虚数部分を参照するためのポインタです。ですからさらにメモリを確保する必要はありません。

MatlabVarName は転送されるデータの MATLAB 変数名が格納されている文字列です。受信データの型は double です。MATLAB 変数は 2 次元マトリックス、ベクトル、あるいはスカラのうちのいずれかです。

MatlabVarName の型、ディメンション、サイズを調べるには、"MIIsReal"、"MIGetNRows"、"MIGetNCols" を使用してください。

成功すれば true を、そうでなければ false を返します。

例：

```
/* MATLAB 変数 EigenValue を、*/
/* 型をチェックしてから受信します。*/

double *inReal, *inImag;

MIExecute("EigenValue = eig[1 2; -3 4])";);

if (MIIsComplex("EigenValue"))
MIGetComplexResult(&inReal, &inImag, "EigenValue");
else
printf("Referenced variable is not
complex!¥n");
```

参照：

→ "MIIsComplex"

5.4.6 MIGetComplexMatrix

構文：

```
bool MIGetComplexMatrix(double *inReal,
                        double *inImag,
                        char *MatlabVarName )
```

引数：

inReal	double 型へのポインタ
inImag	double 型へのポインタ
MatlabVarName	文字列

機能：

MATLAB ワークスペース から ASCET 環境へ、複素数型マトリックスを転送します。

inReal と inImag は、受信データを参照するために使用される 2 つの double 型データへのポインタです。参照する ASCET マトリックスがまだ宣言されていない場合、この関数を実行するためには、MICreateDouble を実行して受信データ用に十分なメモリを確保しておく必要があります。

MatlabVarName は、転送されるデータの MATLAB 変数名が格納されている文字列です。この MATLAB 変数は適切なサイズの 2 次元マトリックスでなければなりません。in の型、ディメンション、サイズを調べるには、"MIIsComplex"、"MIGetNRows"、"MIGetNCols" を使用してください。

成功すれば true を、そうでなければ false を返します。

例：

例 1：

```
/* asdMatrixReal、asdMatrixImag は、 */
/* C コードクラスエレメントリスト内で宣言されている、*/
/* ユーザー定義の ASCET 変数です。*/

MIExecute("[A, v] = eig([1 2; -3 4]);");
MIGetComplexMatrix(asdMatrixReal, asdMatrixImag, "A");
```

例 2：

```
/* inReal および inImag は */
/* ユーザー定義のローカル変数で、 */
/* ASCET で見ることはできません。*/
/* ですから、ユーザーはそのためのメモリを */
/* 確保する必要があります！ */

double *inReal, *inImag;
MICreateDouble(inReal, 4);
MICreateDouble(inImag, 4);

MIExecute("[A, v] = eig([1 2; -3 4]);");
MIGetComplexMatrix(inReal, inImag, "A");

MIDestructDouble(inReal);
MIDestructDouble(inImag);
```

参照：

→ "MIGetMatrix"

→ "MIGetComplexVector"

→ "MIGetComplexScalar"

5.4.7 MIGetComplexVector

構文：

```
bool MIGetComplexVector(double *inReal,
                        double *inImag,
                        char *MatlabVarName )
```

引数：

inReal	double 型へのポインタ
inImag	double 型へのポインタ
MatlabVarName	文字列

機能：

MATLAB ワークスペース から ASCET 環境へ、複素数型ベクトルを転送します。

inReal と inImag は、受信データを参照するために使用される 2 つの double 型データへのポインタです。参照する ASCET 配列がまだ宣言されていない場合、この関数を実行するためには、MICreateDouble を実行して受信データ用に十分なメモリを確保しておく必要があります。

MatlabVarName は、転送されるデータの MATLAB 変数名が格納されている文字列です。この MATLAB 変数は適切なサイズのベクトルでなければなりません。in の型、ディメンション、サイズを調べるには、"MIIsComplex"、"MIGetNRows"、"MIGetNCols" を使用してください。

成功すれば true を、そうでなければ false を返します。

例：

例 1：

```
/* asdVectorReal、asdVectorImag は、C コードクラスの */
/* エレメントリスト内で宣言されている、*/
/* ユーザー定義の ASCET 変数です。*/

MIExecute("A = eig([1 2; -3 4]);");
MIGetComplexVector(asdVectorReal,
```

```
                        asdVectorImag,"A");
```

例 2：

```
/* inReal および inImag は */
/* ユーザー定義のローカル変数で、*/
/* ASCET で見ることはできません。*/
/* そこで、ユーザーはそのためのメモリを */
/* 確保する必要があります！ */

double *inReal, *inImag;
MICreateDouble(inReal, 2);
MICreateDouble(inImag, 2);

MIExecute("A = eig([1 2; -3 4]);");
MIGetComplexVector(inReal, inImag, "A");

MIDestructDouble(inReal);
MIDestructDouble(inImag);
```

参照：

→ "MIGetVector"

→ "MIGetComplexMatrix"

→ "MIGetComplexScalar"

5.4.8 MIGetComplexScalar

構文：

```
bool MIGetComplexScalar(double *inReal,
                        double *inImag,
                        char *MatlabVarName )
```

引数：

`inReal`	double 型へのポインタ
`inImag`	double 型へのポインタ
`MatlabVarName`	文字列

機能：

MATLAB ワークスペース からユーザー環境へ、複素数型スカラデータを転送できます。

`inReal` と `inImag` は、受信データを参照するために使用される 2 つの double 型データへのポインタです。

`MatlabVarName` は、転送されるデータの MATLAB 変数名が格納されている文字列です。この MATLAB 変数はスカラでなければなりません。型を調べるには `MIIsComplex` を使用してください。

成功すれば `true` を、そうでなければ `false` を返します。

例：

例 1：

```
/* asdScalarReal、asdScalarImag は、*/
/* C コードクラスのエレメントリスト内で宣言されている、*/
/* ユーザー定義の ASCET 変数です。*/

MIExecute("v = 3 - 4i;");
MIGetComplexScalar(  &asdScalarReal,
                     &asdScalarImag, "v");
```

例 2：

```
/* inReal および inImag は */
/* ユーザー定義のローカル変数で、*/
/* ASCET で見ることはできません。*/
/* スカラなので、ユーザーは明示的にメモリを */
/* 確保する必要はありません！ */

double inReal;
double inImag;

MIExecute("v = 3 - 4i;");
```

```
MIGetComplexScalar(&inReal, &inImag, "v");
```

参照：

→ "MIGetScalar"

→ "MIGetComplexMatrix"

→ "MIGetComplexVector"

5.4.9 MIGetString

構文：

```
bool MIGetString(  char *destBuffer,
                   char *MatlabVarName )
```

引数：

destBuffer	MATLAB 文字列を格納するための文字列バッファ
MatlabVarName	文字列が格納されている MATLAB 変数の名前

機能：

MIGetString を使えば、MATLAB ワークスペース から文字列を受け取ることができます。データはバッファ destBuffer に格納されます。

成功すれば true を、そうでなければ false を返します。

例：

```
char Buffer[2048];

MIExecute("aString = 'Hello world!'");

MIGetString(Buffer, "aString");
```

参照：

→ "MIIsString"

5.5 データ属性のチェック

本項で紹介する関数を使用して、MATLAB ワークスペース 内の変数のサイズと型を調べることができます。

5.5.1 MIGetNCols

構文：

```
int MIGetNCols( char *MatlabVarName )
```

引数：

MatlabVarName	MATLAB 変数の名前が格納されている文字列

機能：

MatlabVarName で指定された MATLAB 構造体（マトリックス、ベクトル、あるいはスカラ）の列数を返します。

例：

```
int nc;

nc = MIGetNCols("Fibo");
```

参照：

→ "MIGetNRows"

5.5.2 MIGetNRows

構文：

```
int MIGetNRows( char *MatlabVarName )
```

引数：

MatlabVarName	MATLAB 変数の名前が格納されている文字列

機能：

MatlabVarName で指定された MATLAB 構造体（マトリックス、ベクトル、あるいはスカラ）の行数を返します。

例：

```
int nr;


nr = MIGetNRows("Fibo");
```

参照：

→ "MIGetNCols"

5.5.3 MIIsComplex

構文：

```
bool MIIsComplex( char *MatlabVarName )
```

引数：

MatlabVarName	MATLAB 変数の名前が格納されている文字列

機能：

MATLAB 変数が複素数なら true を、そうでなければ false を返します。

例：

```
double *inReal, *inImag;

if (MIIsComplex("Matrix"))
   MIGetComplexResult(&inReal, &inImag,
                      "Matrix");
else
        MIGetResult(&inReal, "Matrix");
```

参照：

→ "MIIsReal"

→ "MIIsEmpty"

→ "MIIsString"

5.5.4 MIIsEmpty

構文：

```
bool MIIsEmpty( char *MatlabVarName )
```

引数：

`MatlabVarName` MATLAB 変数の名前が格納されている文字列

機能：

MATLAB 変数が空なら `true` を、そうでなければ `false` を返します。

注記

空（[ ]）の MATLAB 変数とは、存在しているがまだ値が代入されていない MATLAB 変数のことです。

例：

```
double *in;

if ( ! MIIsEmpty("Matrix") )
   MIGetResult(&in, "Matrix");
```

参照：

→ `"MIIsReal"`

→ `"MIIsComplex"`

→ `"MIIsString"`

5.5.5 MIIsReal

構文：

```
bool MIIsReal( char *MatlabVarName )
```

引数：

`MatlabVarName` MATLAB 変数の名前が格納されている文字列

機能：

MATLAB 変数が実数なら `true` を、そうでなければ `false` を返します。

例：

```
double *inReal, *inImag;

if (MIIsReal("Matrix"))
   MIGetResult(&inReal, "Matrix");
```

```
else
        MIGetComplexResult(&inReal, &inImag,
        "Matrix");
```

参照：

→ "MIIsComplex"

→ "MIIsEmpty"

→ "MIIsString"

5.5.6 MIIsString

構文：

```
bool MIIsString( char *MatlabVarName )
```

引数：

MatlabVarName	MATLAB 変数の名前が格納されている文字列

機能：

MATLAB 変数が文字列なら true を、そうでなければ false を返します。

例：

```
char in[1024];

if (MIIsString("A"))
   MIGetString(in, "A");
else
   printf("Referenced variable is not a character
          string!");
```

参照：

→ "MIIsComplex"

→ "MIIsEmpty"

→ "MIIsReal"

5.6 エラー処理

本項では、MATLAB ワークスペース 内で発生するエラーに対処するために MIP が内部的に使用する関数を紹介します。MIP には MATLAB エラーに対する処理がすべて含まれていますが、以下の関数はユーザーが直接呼び出すためのものではありません。ですから本項は、これらの関数の使用方法を説明するのではなく、エラーに対する内部処理を理解していただくために記述されています。

MATLAB エンジンにはエラー処理ルーチンが用意されていないので、MATLAB エンジンの関数を使用して、専用のルーチン群が開発されました。MATLAB には、ユーザーコールに起因する最新のエラーについてのメッセージ文字列を返す、`lasterr` という関数があります。原則として、MATLAB エラーが発生するのは、MIP 関数 `MIExecute`、`MIExecuteAndEcho`、`MIExecuteWithoutErrCheck` のいずれかをコールした場合のみです。

これらの MIP 関数のうち、`MIExecute` および `MIExecuteAndEcho` のみが `checkMatlabError` というメインエラー処理ルーチンを使用します。発行される `lasterr` エラーメッセージにアクセスするには、`MIExecute` の機能が必要です。エラーチェックが無限に繰り返されるのを防ぐために、`MIExecuteWithoutErrCheck` が実行され、エラーが発生したかどうかを `MIErrorStatus` 内で調べ、エラーがある場合には、MATLAB ワークスペース 内の一時変数にエラーメッセージ文字列を代入してから、`MIGetLastError` 内で `MIGetString` コールを使用してその一時変数を MIP 環境に転送します。その後、不正なエラーメッセージが出力されないように、MATLAB 内では `lasterr` メッセージがリセットされて空の文字列になります。

以上のことから、MIP の通常の使用においては、`checkMatlabError` や `MIExecuteWithoutErrCheck` を直接呼び出しても意味がないことがわかります。

5.6.1 `checkMatlabError`

構文：

```
void checkMatlabError( void )
```

引数：

なし。

機能：

このメインエラーチェック／処理ルーチンは、`MIExecute` 内部で起動され、MATLAB 環境内でエラーが発生したかどうかを調べます。エラーメッセージが ASCET Target Server のエラー例外処理に渡され、MATLAB の `lasterr` がリセットされます。

この関数を直接使用することはお勧めできません。通常、エラーのチェックと処理は、内部で行われます。

参照：

→ "MIGetLastError"

→ "MIErrorStatus"

5.6.2 MIErrorStatus

構文：

```
bool MIErrorStatus( void )
```

引数：

なし。

機能：

MATLAB ワークスペース 内でエラーが発生している場合には true を返します。

この関数は内部で使用するためのもので、ユーザーが直接使用することはお勧めできません。通常、エラー処理は内部で行われます。

参照：

→ "checkMatlabError"

→ "MIGetLastError"

5.6.3 MIGetLastError

構文：

```
bool MIGetLastError( char *buffer )
```

引数：

buffer	文字バッファ

機能：

最新の MATLAB エラーメッセージが buffer に書き込まれます。

この関数は内部で使用するためのものです。通常、エラー処理は内部で行われるので、ユーザーがこの関数を直接使用することはお勧めできません。

参照：

→ "MIErrorStatus"

→ "checkMatlabError"

5.7 内部関数

本項に含まれる関数は、内部で使用するためだけに作られたものであるため、ここでは簡単に概要を記述します。

5.7.1 MIActivateBuffer

構文：

```
bool MIActivateBuffer(char *Buffer,
                      int buflen )
```

引数：

Buffer	文字バッファ
buflen	Buffer の長さ

機能：

Buffer に指定されたバッファを起動し、MIExecute などのファンクションコールの結果として得られる MATLAB 出力を取り込みます。

バッファは、MIExecute、MIExecuteAndEcho、または MIExecuteWithoutErrCheck の次回の実行で、自動的にクリアされます。

ユーザーがこの関数を直接使用することはお勧めできません。直接使用した場合、それが MIExecuteWithoutErrCheck の前に実行された場合に**限り**、空でないバッファが得られます。

MIExecute が実行された場合、内部でエラー処理が行われるので、MATLAB 出力はバッファに取り込まれません。MATLAB ワークスペース の出力にアクセスするには、MIExecuteAndEcho を使用してください。

参照：

→ "MIExecuteAndEcho"

5.7.2 miAsc2Char

構文：

```
bool miAsc2Chars(  char *destination,
                   double *source,
                   int NOfElements )
```

引数：

`destination`	エコー先の文字列のポインタ
`source`	`double` 型配列へのポインタ

機能：

この関数は、文字列転送のために `MIGetString` 内部で使用され、MATLAB 文字列の ASCII コードが格納されている `double` 型配列を受信します。この配列はさらに C 文字列に再構築され、`*destination` に代入されます。

ユーザーがこの関数を直接使用することはお勧めできません。

参照：

→ `"MIGetString"`

エラーメッセージ	発行元関数	考えられる原因
Cannot start MATLAB	`MIOpen`	• MATLAB 6.5 以降のバージョンがインストールされていますか？ • MATLAB は正しくインストールされていますか？ • 正しいプラットフォームを使用していますか？（MIP は Windows 3.11 以前では実行できません。）
Closing MATLAB engine failed. Maybe connection was already closed.	`MIClose`	• すでに `MIClose` で MATLAB を閉じていませんか（あるいは MATLAB ワークスペース が終了してしまっていませんか）？
Retrieving data failed. Referenced variable either does not exist, or is not of type `DOUBLE ARRAY`.	`MIGet` ファミリ、`MIGetNRows`、`MIGetNCols`、`MIIsComplex`、`MIIsReal`、`MIIsEmpty`、`MIGetString`	• 参照されている MATLAB 変数が存在しないか、数値ではありません。 • 使用できるのは、スカラ、ベクトルおよび 2 次元マトリックスのみです。 • MATLAB 変数が文字列ではありませんか？（`MIIsString`、`MIGetString` を使用してください。）
No connection to MATLAB	`MIExecute` ファミリ	• `MIOpen` を忘れていませんか？ • 接続が失われているか、MATLAB がクラッシュしていることも考えられます。
MATLAB variable referred to is not of complex type	`MIGetComplex` ファミリ	• データは複素数ではありません。`MIIsComplex` で確認することをお勧めします。
Put operation failed	`MIPut` ファミリ	• ASCET データを MATLAB に転送することはできません。 • `MIOpen` を忘れていませんか？
An error has occurred inside MATLAB	`MIExecute` ファミリ	• MATLAB コードに誤りがあります。（MATLAB エラーメッセージを参照してください。） MATLAB ワークスペース 内で先にコードを調べてから、ASCET にインストールすることをお勧めします。

MATLAB Integration Package

Simulink® インターフェース

7 Simulink¨インターフェース

Simulink インターフェースによって、Simulink で作成されたモデルを ASCET にインポートすることができます。Simulink モデルをインポートするには、Real-Time Workshop® を使用してモデルから C コードを生成し、そのコードを ASCET にモジュールとして組み込みます。モデルの実行に必要なタスクやプロセスは、ASCET で自動生成されます。

サポートされている MATLAB バージョン

Simulink インターフェースは、以下のバージョンの MATLAB をサポートしています。

- MATLAB R13.1：
  MATLAB 6.5, Simulink 5.1, Real-Time Workshop 5.1,
  Stateflow 5.1 + Stateflow Coder 5.1（オプション）
- MATLAB R14.3：
  MATLAB 7.1, Simulink 6.3, Real-Time Workshop 6.3,
  Stateflow 6.3 + Stateflow Coder 6.3（オプション）

上記の各プログラムは、モデルのインポート時に使用されます。ASCET でモデルをシミュレートする際には、MATLAB、Simulink、RealTime Workshop は必要ありません。

7.1 概要

以降の各項の記述は、Simulink モデルを ASCET にインポートして使用することを目標とするユーザーを対象としています。ここでは Simulink と Real-Time Workshop （RTW) に関する充分な知識と、ASCET と MATLAB での作業の経験が必要です。

7.2 システムアーキテクチャ

図 7-1 は、Simulink インターフェースのシステムアーキテクチャを簡単に示したものです。ASCET と MATLAB/Simulink の間の通信は ASCET の自動化インターフェースを通じて行われるので、ユーザーがこれを意識することはありません。用途に応じて、MATLAB/Simulink が ASCET を、あるいは ASCET が MATLAB/Simulink をリモート制御します。

Simulink モデルがインポートされるときには、実際のモデルファイルとそれに関連する S ファンクションが Simulink に読み込まれ、Real-Time Workshop がそれを C コードに変換します。このコード生成には、スクリプトによって制御される Target Language Compiler が使用され、このスクリプトの中に ASCET への組み込みに必要な情報が含まれています。そしてこの C コードから、Simulink 統合パッケージによって ASCET モジュールが生成され、この過程は、ETAS 固有のコンフィギュレーションファイルによってコントロールされます。

インポートが完了したモデルは、Simulink モデルをカプセル化した 1 つの ASCET モジュールとなり、このモジュールが既存の ASCET プロジェクトに組み込まれます。このモジュールに必要なタスクやプロセスは自動的に生成され、適切に設定されます。

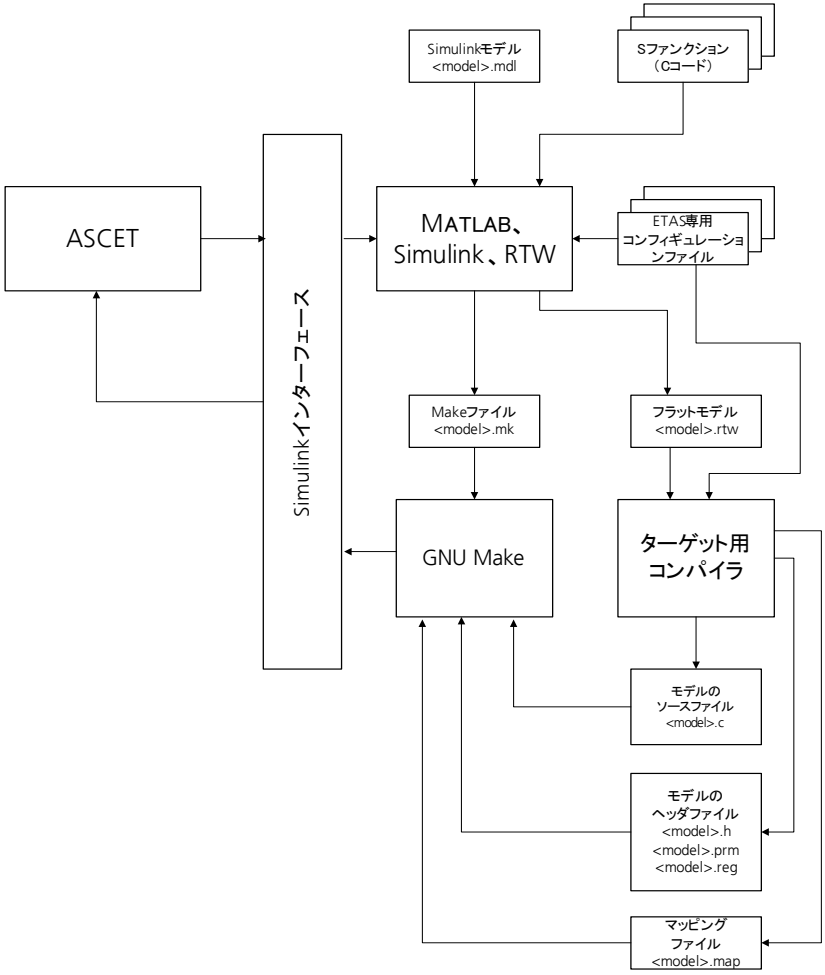


図 7-1　Simulink インターフェースのシステムアーキテクチャ

7.3 インポートされた Simulink モデルの内容

Simulink インターフェースによって生成される ASCET モジュールには、Simulink モデルの全体と Simulink 数値演算ルーチンが、C コードの形式で含まれます。モジュール名には、Simulink モデルのファイル名が使用されます（たとえばファイル名が `gas_eng.mdl` の場合、モジュール名は `gas_eng`）。

本章では、Simulink モデルをインポートしたときに生成されるすべての ASCET データベースアイテムについて説明します。

7.3.1 モデル変数のマッピング

各 Simulink 変数は、それぞれ個別の ASCET 変数にマッピングされます。下の表は、その対応関係をまとめたものです。

Simulink 変数	**ASCET 変数**
幅が 1 である、 最上位モデルレベルの入力ポートブロック	受信メッセージ
幅が 1 である、 最上位モデルレベルの出力ポートブロック	送信メッセージ
幅が 1 より大きい、 最上位モデルレベルの入力ポートブロック	複数の受信メッセージ
幅が 1 より大きい、 最上位モデルレベルの出力ポートブロック	複数の送信メッセージ
スカラパラメータ	パラメータ
ベクトルパラメータ	配列パラメータ
行列パラメータ	行列パラメータ
スカラの連続または離散ステート	変数
ベクトルの連続または離散ステート	配列変数
スカラのブロック出力ポート	変数
ベクトルのブロック出力ポート	配列変数

最上位モデルレベル（ルートレベル）の入力ポートブロックと出力ポートブロックだけが ASCET の送信メッセージと受信メッセージにマッピングされます。幅が 1 より大きい入力ポートブロックおよび出力ポートブロックは、複数の送信メッセージと受信メッセージにマッピングされます。

7.3.2 プロセス

モデルのインポート時に、C 言語で実装された以下のプロセスが ASCET モジュール内に生成されます。

プロセス	説明
`init`	モデルと積分処理を初期化します。
`step`	シミュレーションの 1 ステップの演算を実行します。
`exit`	モデルを終了し、その作業メモリを解放します。
`update_paras`	ASCET パラメータを Simulink パラメータにコピーします。
`update_states`	Simulink のステート変数とブロックの出力ポートを ASCET 変数にコピーします。

シミュレーションの 1 ステップの演算が実行される時には、まず ASCET の受信メッセージが Simulink の入力ポートブロックの信号にコピーされてから、実際の演算処理が実行されます。そして、Simulink の出力ポートブロックの信号が ASCET の送信メッセージにコピーされます。

インポートされた Simulink モデル内のパラメータを適合する場合は、`update_paras` プロセスが必要です。このプロセスが実行されるたびに、ASCET 実験環境で変更された内容がすべて Simulink モデルに転送されます。

また、インポートされた Simulink モデルのステート変数の値とブロックの出力ポート信号を画面表示するには、`update_states` プロセスが必要です。このプロセスが実行されると、対応する ASCET 変数の値が更新されます。

このように、`update_paras` プロセスと `update_states` プロセスは、インポートされた Simulink モデルの実行中に測定と適合を行う場合に限り必要です。このような Simulink プロセスの実行には時間がかかるので、他のモデル（ステッププロセス）よりも大きいインターバルで実行されるようにしてください。

図 7-2 は、各種変数とそれらを処理するプロセスとの関係を示しています。

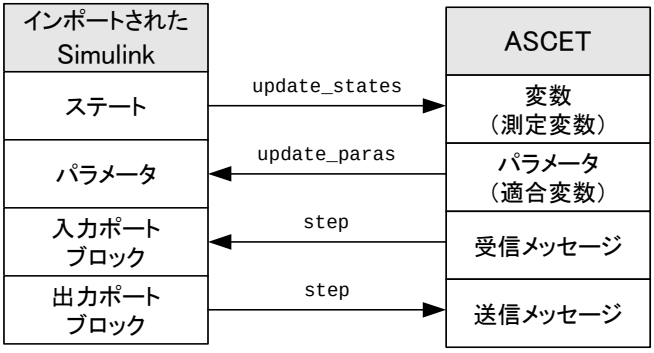


図 7-2　データとプロセス

7.3.3 タスク

前述の各プロセスを実行するために必要なタスクは、Simulink 統合機能により自動的に生成され、ASCET プロジェクトエディタの“OS”タブで内容を確認することができます。各タスクには、必要なプロセスが自動的に割り当てられます。

タスク	プロセス	設定
`sim_init`	`init`	init トリガ、active モード
`sim_step`	`step`	timer トリガ、active モード
`sim_exit`	`exit`	init トリガ、inactive モード

`update_paras` プロセスと `update_states` プロセスについては、タスクは生成されません。モデルのサイズによってはこれらのプロセスの実行に非常に長い時間がかかる場合があるので、これらのプロセスの実行タイミングはユーザーが指定します。通常、これら 2 つのプロセスは、`step` タスクよりも大きなインターバルで実行されるようにしてください。`sim_step` タスクの周期は Simulink モデル内で定義されたステップサイズに相当します。

注記

タスク名は、Real-Time Workshop のオプション設定によって変更することができます。タスク名が変更された後に Simulink モデルが再度インポートされた場合、以前のタスクとプロセスは自動的に削除されません。それらはマニュアル操作で削除してください。
このことは、Simulink モデルが ASCET プロジェクトエディタの“Elements”ペーンで削除された場合にもあてはまります。

7.3.4 クラス

モデルのインポート時には、非常に多くのクラスが ASCET データベース内に生成され、これらのクラスが、生成されたモジュールのプロセスから参照されます。クラスの数と名前はインポートされたモデルにより異なりますが、以下のクラスは必ず生成されます。

クラス	説明
`<model>_<model>`	モデルの初期化と演算のための Simulink コードが含まれています。
`<model>_es_main`	Simulink 積分処理の初期化とシミュレーションステップの演算実行のための Simulink コードが含まれています。

7.3.5 生成される変数の名前

Simulink モデルの各変数に対応して、ASCET 変数が 1 つずつ生成されます。ASCET で生成される変数の名前は、パス名と変数名の 2 つの部分で構成されています。このうち、変数名の部分は、Real-Time Workshop で生成された C コード内

の変数名で、パス名の部分は、その変数が定義されている Simulink ブロックのパスです。これら 2 つの部分の間はアンダースコア 2 文字（“__”）で区切られます。

生成される変数の例

例として、`Vehicle_a` という Simulink モデルに `Engine` というブロックが含まれていて、そのブロックの中には `Cylinder` というブロックが、そしてさらにその中に `n_engine` という変数が含まれていると仮定します。この場合、変数 `n_engine` のパスは `Vehicle_a/Engine/Cylinder` となります。また、Real-Time Workshop では、`n_engine` のブロック出力用として `s1_n_engine` というような内部名を生成します。その結果、ASCET 変数の名前は以下のようになります。

```
Vehiclea_Engine_Cylinder__s1_n_engine_B.
```

この名前においては、各ブロック名はアンダースコアで区切られています。また RTW が生成する内部名は、2 文字分のアンダースコア“__”によってブロックパスの部分と区切られています。サフィックスの `_B` は、この変数がブロック出力であることを示しています。

注記

変数名が生成される際、ブロックパスの各エレメント名に含まれる英数字以外の文字（たとえば `Vehicle_a` 内の“_”）は削除されます。

RTW オプションダイアログ（73 ページの表 7-1 を参照してください）において **Reverse Quantity Names** オプションがオンになっている場合、ASCET の変数名は逆の順番（`<variable name>__<path name>`）で生成されるので、前述の変数の例では、以下のようになります。

```
s1_n_engine_B__Vehiclea_Engine_Cylinder
```

変数のタイプにより、以下のいずれかのサフィックス（接尾辞）が変数名に付加されます。

変数タイプ	サフィックス
パラメータ	`_P`
連続ステート変数	`_C`
離散ステート変数	`_D`
ブロック出力	`_B`

幅 が 1 より大きい入力ポート／出力ポートブロック用には、ASCET 内で複数の変数が生成されます。このようなブロックの Simulink 名には、プレフィックスが付きません。たとえば、`out` という名前の出力ポートブロックについては、変数 `out_01`、`out_02`、`out_3`... が生成されます。

生成される変数の名前の長さは、最長で 128 文字です。各変数の長さの合計がこれを超える場合、各変数名は自動的に省略されて短くなります。

7.4 モデルのインポート

この項では、以下の内容について説明します。

- モデルインポートの実行（69 ページ）
- インポートしたモデルの扱い（80 ページ）

注意事項

モデルのインポートを行う際は、以下の点に注意してください。

- ブロック名、出力名、パラメータ名に使用できる文字は、英数文字とアンダースコアのみです。他にいくつか使用できる特殊文字もありますが、それらによってエラーが発生する場合があります。
- “signal names”という出力を定義しておいてください。
- エラーが発生した場合は、grt_malloc.tlc コード生成が実行できるかを確認してください。エラーが解消されない場合、エラーの原因についての詳しい情報が表示されます。
- `user.*` ファイルの保存を行う必要があるため、モデルは書込み禁止にしないでください。

モデルインポートの実行

モデルのインポート処理は、以下に説明するいくつかのステップに分かれます。

- RTW オプションを設定する（R13 の場合）：（69 ページ）
- RTW オプションを設定する（R14 の場合）：（71 ページ）
- Simulink モデルをインポートする：（75 ページ）
- 検索パスを指定する：（77 ページ）
- インポートされたモデルをセットアップする：（78 ページ）
- 複数の Simulink モデルのインポート（78 ページ）

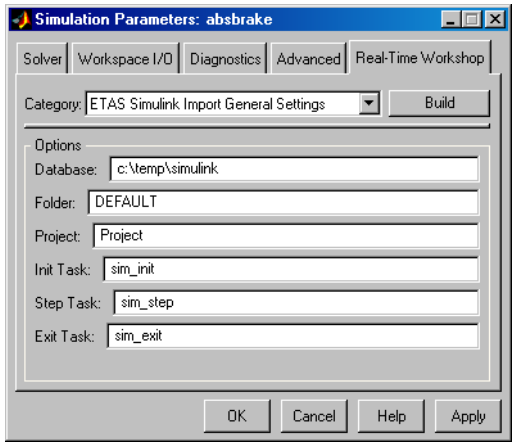
RTW オプションを設定する（R13 の場合）：

- Simulink モデルを開きます。
- **Simulink から Tools → Real-Time Workshop → Options** を選択します。

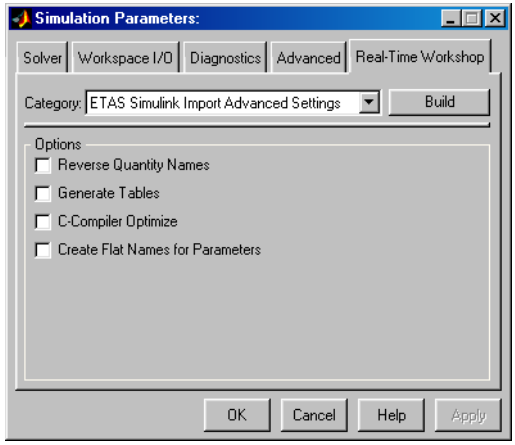
  “Simulation Parameters”ダイアログボックスが開きます。
- “Real-Time Workshop”タブを選択します。
- “System target file”フィールドに ETAS ターゲット `etas.tlc` を入力します。

  Make コマンドや Make ファイルテンプレートの値が自動的に挿入されます。

- “Category”コンボボックスから、ETAS Simulink Import General Settings というエントリを選択します。



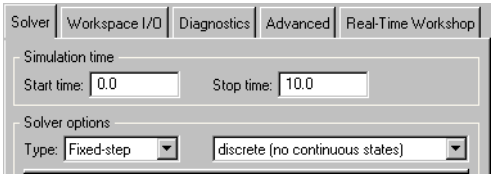
- オプション設定を行います。
- “Category”コンボボックスから、ETAS Simulink Import Advanced Settings というエントリを選択します。



- オプション設定を行います。

  各オプションについては、73 ページの表 7-1 を参照してください。

- “Solver”タブを選択します。



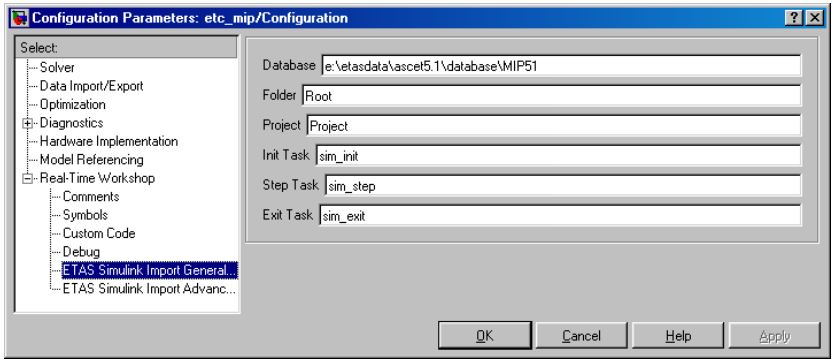
- “Solver Options”フィールドで `Fixed-step` を選択し、さらに適切なソルバを選択します。
- **すべての設定が終わったら、OK** をクリックします。

RTW オプションを設定する（R14 の場合）：

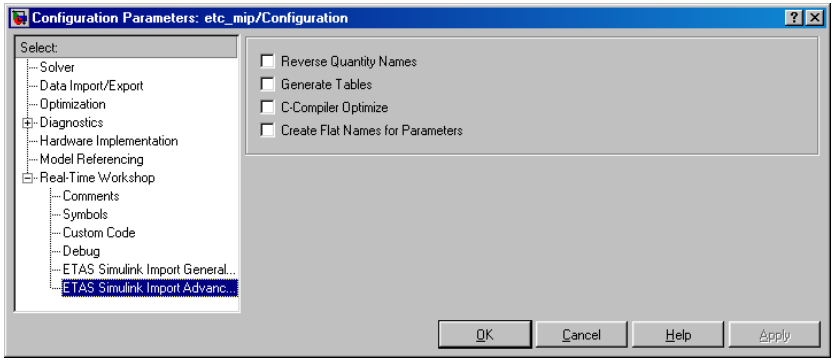
- Simulink モデルを起動します。
- **Simulink から Simulation → Configuration Parameters** を選択します。

  “Cofiguration Parameters”ダイアログボックスが開きます。
- “Cofiguration Parameters”ダイアログボックスの“Select”ペーンで、`Real-Time Workshop` というエントリを開きます。
- “System target file”フィールドに ETAS ターゲット `etas.tlc` を入力します。

  Make コマンドや Make ファイルテンプレートの値が自動的に挿入されます。
- `Real-Time Workshop` エントリを展開して `ETAS Simulink Import General Settings` というエントリを開きます。

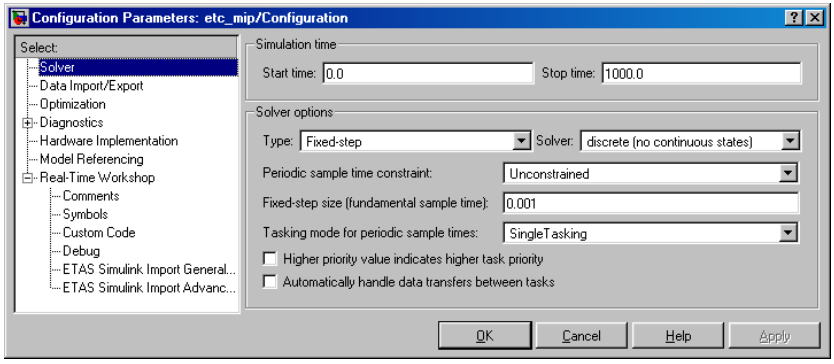


- オプション設定を行います。
- “Select”ペーンの `ETAS Simulink Import Advanced Settings` というエントリを開きます。



- オプション設定を行います。

  各オプションについては、73 ページの表 7-1 を参照してください。
- “Select”ペーンの `Solver` というエントリを開きます。



- “Solver Options”フィールドで `Fixed-step` を選択し、さらに適切なソルバを選択します。
- **すべての設定が終わったら、OK** をクリックします。

オプション	意味
Database	すでにインポートされているモデルが格納されている ASCET データベースです。
Folder	すでにインポートされているモデルが格納されている、ASCET データベースのフォルダです。
Project	Simulink モデルが格納されている ASCET プロジェクトです。
[...] Task Name	Simulink インターフェースによって作成されるタスクの名前です。詳細は、67 ページの「タスク」という項を参照してください。
Reverse Quantity Names	変数名とパス名の位置が逆になっている変数名を生成します。詳細は、67 ページの「生成される変数の名前」という項を参照してください。
Generate Tables	Simulink テーブルを ASCET テーブルにマッピングします。それ以外の場合は、ASCET 配列やマトリックスが生成されます。
C-Compiler Optimize	ES113x で使用される DIAB データコンパイラは、デフォルト設定においては最適化を行いません。このオプションがオンになっていると最適化が行われます。
Generate Flat Names For Parameters	モデルのすべてのパラメータがワークスペース変数として定義されている場合にのみ、このオプションをオンにします。

表 7-1　コード生成オプション

上記以外にもいくつかのオプションがあります。これらのオプションは ETAS 固有のものではないので、Real-Time Workshop のマニュアルを参照してください。以下に、それらのオプションの中から特に重要なものをあげて簡単に説明します。

- **Show eliminated statements（R13、R14）**

  コード生成に必要ないステートメントがコメントになります。

- **Loop rolling threshold（R13）/ Loop unrolling threshold （R14）**

  基本ブロックの入出力用の配列を持つすべてのエレメントについて、同じ設定が行われる必要があります。配列長がこのしきい値よりも長くなると、for ループが生成されます。そして同様の配列を持つ各エレメントについて同様のコードが生成されます。

- **Verbose builds（R13）/ Verbose build（R14）**

  コード生成の進行状況がメッセージと共に出力されます。

- **Inline invariant signals（R13、R14）**

  接続された入力により常に一定の値となるブロック出力値が、削除されます。

  注記

  このオプションによってコードの効率は向上しますが、この出力値をモニタすることができなくなります。

- **Local block outputs（R13）/ Enable local block outputs（R14）**

  このオプションがオンになっていると出力値用にローカル変数が生成され、オフになっているとグローバル変数が生成されます。

- **Force generation of parameter comments（R13）**
  / Verbose Comments for SimulinkGlobal Storage Class（R14）

  このオプションがオンになっていると、オリジナルのパラメータ名とブロック名を含むパラメータについて、コメントが生成されます。オプションがオフになっていると、モデル内に 1000 以上のパラメータが含まれる場合、コメントは生成されなくなります。パラメータ数が 1000 未満の場合は、このパラメータはコード生成に影響を与えません。

- **Buffer reuse（R13）**

  コード生成時に、各変数ごとに個別のメモリを割り当てず、不要になったメモリを再利用します。

  注記

  警告：この機能により、モニタされる出力値が正しく表示されなくなる場合があります。通常このオプションはオフにしておいてください。

- **Include system hierarchy number in identifiers（R13 のみ）**

  生成される名前の長さに影響します。

- **Expression folding（R13 のみ）**

  “Expression folding”は、複数の式を組み合わせて 1 つの短い式を作成する（つまり「式を畳み込む」）機能です。これによりコードは効率化されますが、これは単純な Simulink 基本ブロックが使用されている場合にのみ適用されます。これによって変更される中間出力も削除されるため、通常はこのオプションはオフにしておいてください。この機能により S ファンクションの各出力値用にそれぞれ変数が生成されるため、モデルに S ファンクションのみが含まれ Simulink 基本ブロックが含まれない場合は、このオプションはコード生成に影響しません。

  注記

  R14 の場合、この「式の畳み込み」機能は複数のオプションでモードを設定する必要があります。詳しくは RTW のユーザーズガイドをお読みください。

- **Fold unrolled vectors（R13 のみ）**

  入出力用の配列を持つ式も畳み込まれます。

- **Enforce integer downcast（R13）**
  / Ignore integer downcasts in folded expressions（R14）

  このオプションは常にオンにしておいてください。オフになっているとオーバフロー（たとえば 32 プロセッサでの 8 ビットまたは 16 ビット演算によるオーバーフロー）が発生する可能性があります。

- **Generate comments（R13）/ Include comments （R14）**

  コード内にコメントを生成します。

“Simulation Parameters”ダイアログの“Advanced”タブでは、Real-Time Workshop のコード生成に影響するオプションを設定でき、これらのオプションを以下に簡単にまとめます。詳しくは Simulink のマニュアルを参照してください。

- **Inline parameters（R13、R14）**

  すべてのパラメータが“inline”として扱われます。つまり、それらの値が定数として使用され、これはつまり、コード内のすべての依存式が必要に応じて単純化されることを意味します。しかしこのようにコード生成されたパラメータの値は、以後の変更ができなくなります。パラメータ値が MATLAB ワークスペースから提供される場合、“Configure...”ボタンを使用して、各パラメータごとに設定を行うことができます。

- **Block reduction（R13）/ Block reduction optimization（R14）**

  “Expression folding”に似た機能です。通常はオフにしておいてください。

- **Boolean logic signals（R13）**
  / Implement Logic Signals as Boolean Data（R14）

  旧バージョンの Simulink では許可されていなかったコードの妥当性をチェックします。旧バージョンのモデルを使用している場合以外は、常にオンにしておいてください。

- **Parameter pooling（R13 のみ）**

  このオプションは、モデルのパラメータが MATLAB ワークスペース内で定義されていて、同一の配列が非常に多く存在している場合にのみ影響します。通常はオフにしておいてください。

- **Signal storage reuse（R13、R14）**

  RTW に対してシグナルメモリの再利用を要求します。“Buffer reuse”と似た機能です。通常はオフにしておいてください。

Simulink モデルをインポートする：

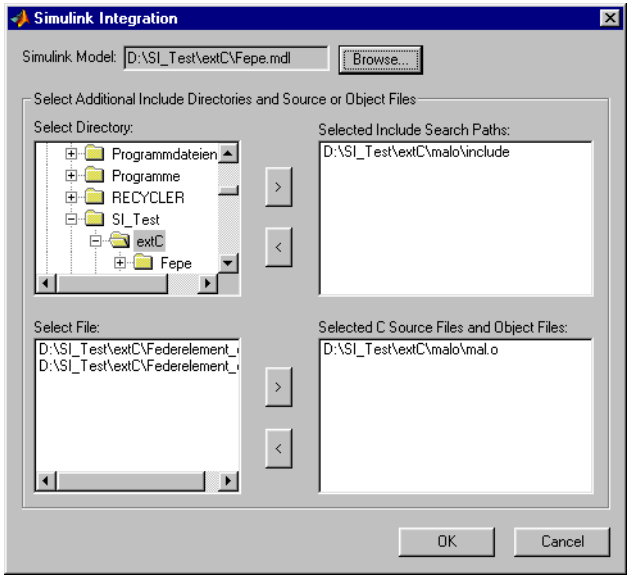
- ASCET のコンポーネントマネージャを開きます。

- 新しいプロジェクトを作成し、プロジェクトエディタで開きます。

または

- プロジェクトエディタで既存のプロジェクトを開きます。
- プロジェクトエディタで **Simulink → Import Model** を選択します。

  “Simulink Integration”ダイアログボックスが開きます。



- インポートする Simulink モデルを選択します。

  “Selected Directory”フィールドに現在のディレクトリツリーが表示されます。
- 外部の C ソースコードやオブジェクトファイルを使用しない単純なモデルの場合、検索パスを指定する必要はありません。
- **OK** をクリックします。

検索パスを指定する：

- 上記の“Simulink Integration”ダイアログの“Select Directory”フィールドでパスを選択して **>** ボタンをクリックすると、そのパスが“Selected Include Search Paths”リストに追加されます。

  コンパイラは、S ファンクションなどによって使用されるインクルードファイルを、これらのパスから検索します。

  選択されたパスに格納されている C ソースとオブジェクトファイルが“Select File”リストに表示されます。

- このリストからファイル（1 つまたは複数）を選んで **>** ボタンをクリックすると、選択されたファイルが“Selected C Source Files and Object Files”リストに追加されます。

  これらのファイルは後にリンクされます。これらは、S ファンクションによって呼び出されるサブルーチンが含まれるものです。

注記

これらのオブジェクトファイルが生成された時のターゲットとコンパイラが、プロジェクトのターゲットとコンパイラに一致している必要があります。

- **OK** をクリックします。

上記の操作によって Simulink モデルがインポートされます。実際には 65 ページの「インポートされた Simulink モデルの内容」に示されているデータ構造体が、ASCET データベース内に生成されます。MATLAB は、モデルのインポート実行時に自動的に起動され、インポート過程の個々のステップが MATLAB コンソールウィンドウに表示されます。この時発生する可能性のあるエラーについての詳細は、83 ページ「トラブルシューティング」の章を参照してください。

モデルインポートが正常に終了した後、オペレーティングシステムおよびグローバル通信に関して以下のような設定を行ってください。

インポートされたモデルをセットアップする：

- `update_states` および `update_paras` プロセスを、必要に応じて任意のタスクに割り当てます。

  これらのプロセスは大きなインターバルで実行されるのが普通なので、自動的には割り当てられません。詳細は、66 ページの「プロセス」という項を参照してください。
- グローバルエレメント（送信メッセージと受信メッセージ）を作成、あるいは名前を変更して Simulink 変数名にあわせ、プロジェクトのグローバル通信をセットアップします。
- **Global Elements → Resolve Globals** を選択して、グローバルエレメントの参照を解決します。

注記

Simulink インターフェースが生成した ASCET 変数（送信／受信メッセージなど）の名前は変更しないでください。変更してしまうと、その後同じモデルを改めてインポートするたびに同じ変更作業を繰り返さなければならなくなります。またプロジェクトに定義されているグローバルエレメントのうち、生成されるエレメントにマッピングされるものの名前は、必ず変更してください。こうしておけば、モデルを再インポートした後も設定を変更する必要がありません。

注記

インポートされた Simulink モデルでオフラインシミュレーションを実行する際は、シミュレーション実行開始前に、イベントジェネレータで“Exit”タスクを無効にしておいてください。これを行わないと、L1 エラーによってシミュレーションが即時に終了してしまいます。

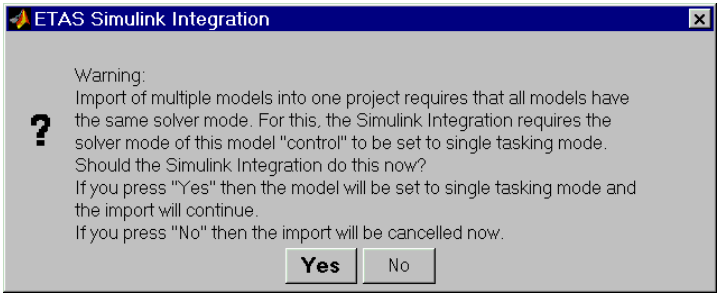
複数の Simulink モデルのインポート

Simulink インターフェースでは、1 つの ASCET プロジェクトに複数の Simulink モデルを連続してインポートすることができます。

この際、以下の点について注意してください。

- 各モデルごとにそれぞれユニークな名前をつけてください。つまり、対応する `.mdl` ファイルの名前の最初の 6 文字が、他のファイルと異なるようにしてください。
- すべてのモデルで同一の積分処理（ソルバ）を使用してください（`ODE1`、`ODE4` など）。

- すべてのモデルで同一の Simulink タスクモデル（`SingleTasking`）を使用してください。
  ソルバモードとして `SingleTasking` が使用されていないと、以下の警告メッセージが表示されます。



  Yes をクリックするとモデルのソルバは `SingleTasking` に設定され、**No** をクリックするとエラーメッセージが表示されます。

- モデルのインポート時に、すべての Simulink モデルが同じ基本ステップサイズを使用していること、またそうでない場合は各シミュレーションステップタスクの名前がそれぞれ異っていることを確認してください。
- モデルパーツが S ファンクションを使用している場合、ある特定の S ファンクションは、インポートされる複数のモデルにおいて同時に使用されていることが許されません。この場合、S ファンクションの名前を変更し、各パーツからそれぞれ異なる名前で呼び出されるようにしてください。
- Simulink の各入力／出力に対して ASCET メッセージが生成されるため、たとえば `ModelA` に `pressure_A` という出力があり、`ModelB` にも `pressure_A` という入力がある場合、ASCET の一般的な規則に従って、これらの入力と出力は暗黙的に接続されます（詳しくは、『ASCET リファレンスガイド』を参照してください）。
- シミュレーションステップタスクは、デフォルトでは `sim_step` という名前です。このデフォルト設定は、Real-Time Workshop のオプション設定で変更することができます。

注記

Simulink における統合プロシージャは、どのような場合でも必ず自動設定によって行うようにしてください。自動設定を変更した場合、インポート中やシミュレーション実行中にエラーが発生します。

注記

モデルの範囲を越えたシーケンシングは、複数の Simulink モデルがインポートされたときに数値的な問題を招く原因となるため、行われません。インポートされたモデルのシーケンシングは、適宜マニュアル操作で調整してください。

7.4.1 インポートしたモデルの扱い

この項には、インポートされたモデルの使用方法に関する以下の項目について説明します。

- ターゲットの変更（80 ページ）
- インポートとコンパイルを異なる ASCET セッション内で行う（80 ページ）
- ファンクションコールサブシステムを ASCET に組み込む（81 ページ）

ターゲットの変更

インポートしたモデルが、たとえば PC ターゲット用のものであった場合、それを実験ターゲット（ES1130 または ES1135）上で実行するには、そのモデルを新しいターゲット用のものとしてコピーする必要があります。

インポートしたモデルを他のターゲット用にコピーする：

- プロジェクトエディタで、コード生成とターゲットについてのオプションを、コピー先とするターゲットにあわせて変更します。
- **Extras → Copy C-Code From** を選択します。
- コピー元とするターゲットを、選択ボックスから選択します。

  モデルをインポートしたときに選択されていたターゲットを選択します。
- **“OS”** タブをクリックします。
- **Operating System → Copy From Target** を選択します。
- モデルをインポートしたときに選択されていたターゲットを選択します。
- **“Files”** タブをクリックします。
- **Project Files → Copy From Target** を選択します。
- モデルをインポートしたときに選択されていたターゲットを選択します。

  これで、モデルのコンパイルと実験が行えるようになりました。

インポートとコンパイルを異なるASCET セッション内で行う

ASCET のオプションダイアログで **Keep files in Generation Directory on Shutdown** オプションがオフになっていると（詳しくは『ASCET ユーザーズガイド』を参照してください）、ASCET のコード生成ディレクトリである .¥cgen¥ 内のファイルは、ASCET をシャットダウンした時に自動的に削除されます。このオプションをオンにすると、次回のセッションからファイルの削除は行われなくなります。

前回のセッション終了時に .¥cgen¥ ディレクトリ内のファイルが削除されていた場合、モデルのコンパイル時に必要なヘッダファイルやコンフィギュレーションファイルはハードディスク上には存在しません。しかしこれらのファイルの内容は ASCET データベース内にプロジェクトと共に保存されているため、いつでもファイルを復元することができます。この作業は、前回の ASCET セッションにおいてインポートされ、まだコンパイルされていないモデルを、今回のセッションでコンパイルする場合に必要です。

プロジェクトファイルを復元する：

- プロジェクトエディタの“Files”タブをクリックします。
- **Project Files → Write all Files** を選択します。

  ファイルが復元されます。これで、前回のセッションでインポートのみを行ったモデルをコンパイルできます。

ファンクションコールサブシステムを ASCET に組み込む

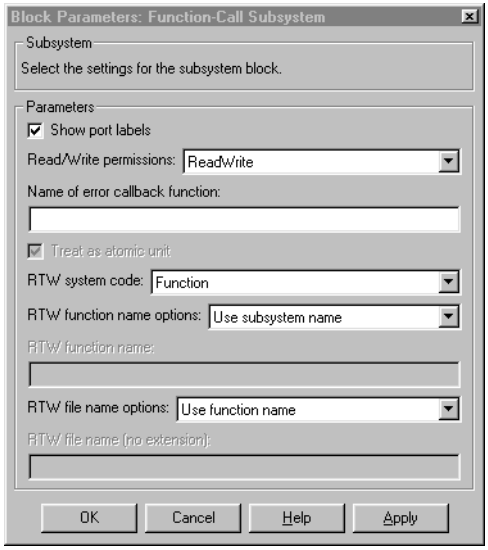
ファンクションコールサブシステムをインポートするには、インポートするブロックについて以下のようにセットアップを行う必要があります。

ファンクションコールサブシステムをセッティングする：

- Simulink モデル内の“Function Call Subsystem”ブロックを右クリックします。
- **Subsystem parameters** を選択します。

  “Block Parameters: Function Call Subsystems！ダイアログ開きます。

- 以下のように設定します。



Simulink インターフェースでファンクションコールサブシステムをインポートする際、そのサブシステム用にいくつかのプロセスが生成されますが、それらのプロセスはタスクリストには組み込まれません。たとえば `model` というモデルに `subsys` という Simulink サブシステムを組み込む場合、以下のような 3 つの ASCET プロセスが生成されます。

- `subsys_initialize_model`
- `subsys_start_model`
- `subsys_model`

サブシステムの組み込みは、以下のように行います。

ファンクションコールサブシステムのプロセスを組み込む：

- トリガの初期化を行う初期化タスクがまだ作成されていない場合は、それを作成します。
- 初期化タスクに `subsys_initialize_model` というプロセスを割り当て、メインの Simulink モデル内の `*_init_*` というプロセスの後に配置します。

- subsys_start_modelというプロセスを初期化タスクに割り当て、subsys_initialize_model プロセスの後に、最後のプロセスとして配置します。

  複数の Triggered サブシステムが存在する場合は、それらの初期化プロセス *_initialize_* は、すべて *_start_* プロセスの前に実行されるようにしてください。
- トリガ用のイベントタスクを作成します。
- Triggered サブシステムから生成された Simulink プロセス、subsys_model をタスクに割り当てます。

注記

上記の処理は、Simulink モデルをインポートするたびに毎回行ってください。

タスクをトリガするイベントは、RTIO エディタなどにおいて、個別に定義する必要があります。

7.5 トラブルシューティング

本項では、Simulink モデルをインポートするときに発生する可能性のあるいくつかの問題とその解決策について説明します。

MATLAB を起動できない

エラーメッセージ： “Fatal Error: Cannot start MATLAB engine.”

原因：MATLAB が PC 上にインストールされていません。

対策：適切なバージョンの MATLAB をインストールしてください。

パス名の空白文字によってインポートエラーが発生する

エラーメッセージ： “Error (execstring): 0 — Unterminated string” and “Error (execstring): -1 — syntax error” （MATLAB コマンドウィンドウに表示されます。）

原因：Simulink は、RTW オプションダイアログにおいて空白文字を含む文字列を扱うことができません。

対策：ASCET データベースへのパスに空白文字が含まれないようにしてください。

Simulink モデルをコンパイルできない

原因 1：ASCET のシャットダウン時に cgen ディレクトリが削除されました。

対策 1：プロジェクトファイルを復元してください。詳細は、80 ページの「インポートとコンパイルを異なる ASCET セッション内で行う」という項を参照してください。

原因 2：Simulink モデルに外部 C コードが含まれています。

対策 2：Make コマンドを修正して、外部 C コードを統合します。詳細は、77 ページの「検索パスを指定する：」という項を参照してください。

オフラインシミュレーションがキャンセルされてしまう

エラーメッセージ：“L1-Message Fatal Error”

原因：ASCET がオフライン実験のイベントリストに Exit タスク `sim_exit` 用のイベントを挿入しました。

対策：イベントジェネレータでこの Exit タスク用のイベントを無効にします。

TCP/IP を正しく設定できない

エラーメッセージ：“Cannot start Tool server on port 32503”

原因 1：ドライバがインストールされていないか、または正しくインストールされていません。PC カードが取り外されている可能性もあります。

対策 1：ネットワークの接続と設定を確認してください。

原因 2：TCP/IP が正しく設定されていません。ループバックアダプタが必要です。

対策 2：システム管理のご担当の方にご相談ください。

8 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。

ヨーロッパ（フランス、ベルギー、ルクセンブルグ、イギリスを除く）

ETAS GmbH

Borsigstrasse 14	Phone:	+49 711 8 96 61-0
70469 Stuttgart	Fax:	+49 711 8 96 61-105
Germany	E-mail:	sales@etas.de
	WWW:	www.etasgroup.com

フランス、ベルギー、ルクセンブルグ

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

イギリス

ETAS Ltd.

Studio 3, Waterside Court	Phone:	+44 1283 - 54 65 12
Third Avenue, Centrum 100	Fax:	+44 1283 - 54 87 67
Burton-upon-Trent	E-mail	sales@etas-uk.net
Staffordshire DE14 2WQ	WWW:	www.etasgroup.com
UK		

USA

ETAS Inc.

3021 Miller Road	Phone:	+1 (888) ETAS INC
Ann Arbor, MI 48103	Fax:	+1 (734) 997-9449
USA	E-mail	sales@etas.us
	WWW:	www.etasgroup.com

日本

イータス株式会社

〒 220-6217	Phone:	(045) 222-0900
神奈川県横浜市西区	Fax:	(045) 222-0956
みなとみらい 2-3-5	E-mail:	sales@etas.co.jp
クイーンズタワー C 17F	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

索引

と

な

は

ふ

へ

も