MATLAB®

The Language of Technical Computing

Computation

Visualization

Programming

The MATH WORKS Inc.

External Interfaces

Version 6　日本語版

サイバネットシステム株式会社への連絡方法

03-5978-5410 Phone

03-5978-5440 Fax

サイバネットシステム株式会社
〒 112-0012 東京都文京区大塚 2 丁目 15 番 6 号
ニッセイ音羽ビル

http://www.cybernet.co.jp/products/matlab/index.htmlWeb

infomatlab@cybernet.co.jp 営業部
techmatlab@cybernet.co.jp 技術部

MATLAB External Interfaces

Printing History:

December 1996	First printing	
July 1997	Revised for 5.1 (online only)	
January 1998	Second printing	Revised for MATLAB 5.2
October 1998	Third printing	Revised for MATLAB 5.3 (Release 11)
November 2000	Fourth printing	Revised and renamed for MATLAB 6.0 (Release 12)
April 2001	翻訳	

Contents

MATLAB からの C および Fortran プログラムの呼び出し

1

2 C 言語 MEX- ファイルの作成

3 Fortran MEX- ファイルの作成

4 C および Fortran プログラムからの MATLAB の呼び出し

5 MATLAB からの Java の呼び出し

6 データの読み込みと書き込み

7 ActiveX と DDE のサポート

8 シリアルポート I/O

1

MATLAB からの C および Fortran プログラムの呼び出し

MATLAB® は、プログラミングとデータ操作を完備した環境をもっていますが、MATLAB 外部のデータやプログラムと相互にデータを交換すると有効な場合があります。MATLAB は、C および Fortran で作成された外部プログラムへのインタフェースを提供します。このインタフェースは、Application Program Interface または、API の一部です。

本章は、つぎのトピックスを説明します。

- MEX- ファイルの紹介
- MATLAB データ
- MEX- ファイルの作成
- MEX- ファイルのカスタムビルド
- トラブルシューティング
- その他の情報

アプリケーションの作成における情報やサポートについては、本章の「その他の情報」部分を参照してください。

注意 ディレクトリパスを参照するプラットフォーム独立の記述について、本マニュアルは UNIX の用法を利用します。たとえば、mex ディレクトリの一般的な参照は、<matlab>/extern/examples/mex です。

MEX- ファイルの紹介

MATLAB から、C または Fortran のサブルーチンを組み込み関数のように呼び出すことができます。MATLAB から呼び出し可能な C および Fortran プログラムは、MEX- ファイルと呼びます。MEX- ファイルは、MATLAB インタプリタが自動的にロードし実行できるダイナミックにリンクされたサブルーチンです。

MEX- ファイルには、つぎのような応用があります。

- 既存の大きい C や Fortran プログラムを M- ファイルとして書き直さずに MATLAB から呼び出すことができます。
- MATLABでは十分な速さで実行しないボトルネックの計算(通常forループ)を、効率を上げるために C や Fortran で記録できます。

MEX- ファイルは、すべてのアプリケーションに対して適切であるわけではありません。MATLAB は、C や Fortran のようなコンパイル言語において、時間のかかる低レベルのプログラミングを削減する特徴をもつ高生産性システムです。一般に、ほとんどのプログラミングは、MATLAB で行われるべきです。アプリケーションで必要としない限り、MEX 機能は使わないでください。

MEX- ファイルの使用法

MEX- ファイルは、C または Fortran ソースコードから生成されるサブルーチンです。これは、M- ファイルや組み込み関数のように動作します。M- ファイルがプラットフォームに依存しない拡張子 .m を持つのに対し、MEX- ファイルはプラットフォーム毎に固有の拡張子を持ちます。つぎの表は、MEX- ファイルに対するプラットフォーム固有の拡張子の一覧です。

表 1-1: MEX- ファイルの拡張子

プラットフォーム	MEX- ファイルの拡張子
Alpha	mexaxp
HP, version 10.20	mexhp7
HP, version 11.x	mexhpux
IBM RS/6000	mexrs6
Linux	mexglx

表 1-1: MEX- ファイルの拡張子

プラットフォーム	MEX- ファイルの拡張子
SGI, SGI64	mexsg
Solaris	mexsol
Windows	dll

MEX- ファイルは、M- ファンクションを呼び出すのと同様に呼び出すことができます。たとえば、ディスク上の MATLAB datafun ディレクトリの conv2.mex という MEX- ファイルは、行列の 2 次元のコンボリューションを行います。conv2.m は、ヘルプテキストのドキュメントのみを含みます。MATLAB 内部から関数 conv2 を呼んだ場合、インタプリタは、MATLAB サーチパス上のディレクトリのリストを検索します。最初に検索される表の対応するファイル名の拡張子、または、.m をもつ conv2 というファイルを各ディレクトリで探します。見つかったら、ファイルをロードして実行します。MEX- ファイルは、同じディレクトリに似た名前のファイルが存在するとき、M- ファイルを優先します。しかし、ヘルプテキストのドキュメントは、.m ファイルから読み込まれます。

接頭語 mx と mex の区別

接頭語 mx をもつ API のルーチンを使って、mxArray の作成、アクセス、操作、破棄が可能です。接頭語 mex をもつルーチンは、MATLAB 環境で実行する操作を行います。

mx ルーチン

配列のアクセスと作成のライブラリは、MATLAB 配列の操作用の配列アクセスおよび作成ルーチンを提供します。これらのサブルーチンは、オンラインの API リファレンスガイドに詳しく説明されており、常に接頭語 mx で始まります。たとえば、mxGetPi は、配列内部で虚数データのポインタを取得します。

配列アクセスおよび作成ライブラリ内のほとんどのルーチンは、MATLAB 配列を操作しますが、IEEE ルーチンとメモリ管理ルーチンの 2 つの例外があります。たとえば、mxGetNaN は、mxArray ではなく、double を出力します。

mex ルーチン

接頭語 mex で始まるルーチンは、MATLAB 環境で実行する操作を行います。たとえば、mexEvalString ルーチンは、MATLAB ワークスペース内の文字列を評価します。

注意　mex ルーチンは、MEX-function 内でのみ利用可能です。

MATLAB データ

MEX- ファイルをプログラミングする前に、MATLAB がサポートする多くのデータタイプを、どのように表現するかを理解する必要があります。この節は、つぎのトピックスを説明します。

- MATLAB 配列
- データストレージ
- MATLAB のデータタイプ
- データタイプの使用

MATLAB 配列

MATLAB言語は、MATLAB配列という単一のオブジェクトタイプでのみ機能します。スカラ、ベクトル、行列、文字列、セル配列、構造体、オブジェクトを含むすべての MATLAB 変数は、MATLAB 配列として格納されます。C では、配列は、タイプ mxArray と宣言されます。mxArray 構造体は、以下のものを含みます。

- タイプ
- 次元
- 配列に対応するデータ
- 数値の場合、変数が実数であるか複素数であるか
- スパース行列の場合、インデックスと非ゼロの最大要素
- 構造体またはオブジェクトの場合、フィールド数とフィールド名

データストレージ

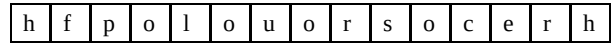
すべての MATLAB データは、列単位で格納されます。これは、Fortran の行列の格納方法です。MATLAB は、オリジナルでは Fortran で書かれているため、この慣例を使います。たとえば、つぎの行列を与えます。

```
a=['house'; 'floor'; 'porch']

a =
house
floor
porch
```

次元は、以下のようになります。

```
size(a)

ans =
   3    5
```

データは以下のように格納されます。



MATLAB のデータタイプ

複素倍精度行列

MATLAB の最も一般的なデータタイプは、複素数倍精度非スパース行列です。これらの行列は、タイプ double で、m 行 n 列です。ここで、m は行数で、n は列数です。データは、倍精度数の 2 つのベクトルとして格納されます。1 つには実数データが含まれ、もう 1 つには虚数データが含まれます。このデータのポインタは、pr（実数データのポインタ）および pi（虚数データのポインタ）として参照されます。実数のみの倍精度行列は、pi が NULL です。

数値行列

MATLAB は、その他のタイプの数値行列もサポートします。これらは、符号付きおよび符号なしの単精度浮動小数点および 8，16，32 ビット整数です。データは、倍精度行列と同じ方法で 2 つのベクトルに格納されます。

MATLAB 文字列

MATLAB 文字列は、タイプ char で、虚数データ成分がないことを除いて、符号なし 16 ビット整数と同じ方法で格納されます。文字列内の各キャラクタは、16 ビット ASCII Unicode として格納されます。C と異なり、MATLAB 文字列は NULL で終了しません。

スパース行列

スパース行列は、MATLAB ではフル行列とは異なるストレージ方法をもちます。パラメータ pr と pi は、倍精度数の配列ですが、nzmax, ir, jc の 3 つの新しいパラメータがあります。

- nzmax は、ir と pr の長さ、また存在すれば、pi の長さを含む整数です。これは、スパース行列の非ゼロ要素数の最大数です。

- ir は、pr と pi の対応する要素の行インデックスを含む長さ nzmax の整数配列を示します。
- jc は、列のインデックスの情報を含む長さが N+1 の整数配列を示します。範囲 0≤j≤N-1 の j に対して、jc[j] は j 列目の最初の非ゼロ要素の ir と pr（存在すれば pi も）のインデックスで、jc[j+1] - 1 は最後の非ゼロ要素のインデックスです。結果として、jc[N] は行列の非ゼロ要素数である nnz と等しくなります。nnz は nzmax より小さければ、追加のストレージを割り当てずに配列内により多くの非ゼロ要素を挿入することが出来ます。

セル配列

セル配列は、mxArray がセルとして参照される MATLAB 配列の集合です。これにより、異なるタイプの MATLAB 配列を一緒に格納することができます。セル配列は、データ部分が mxArray のポインタのベクトルを含むことを除いて、数値配列と同じ方法で格納されます。このベクトルの要素は、セルと呼ばれます。各セルは、それ以外のセル配列を含む、任意のサポートされているデータタイプでかまいません。

構造体

1 行 1 列の構造体は、1 行 n 列のセル配列と同じ方法で格納されます。ここで、n は、構造体のフィールド数です。データベクトルの要素は、フィールドと呼ばれます。各フィールドは、mxArray に格納された名前に関連付けられます。

オブジェクト

オブジェクトは、構造体と同じ方法で格納およびアクセスされます。MATLAB では、オブジェクトは登録されたメソッドをもつ名前が付いた構造体です。MATLAB 以外では、オブジェクトはオブジェクト名を識別する追加のクラス名に対するストレージを含む構造体です。

多次元配列

任意のタイプの MATLAB 配列は、多次元にすることができます。整数のベクトルは、各要素が対応する次元のサイズであるとして格納されます。データのストレージは、行列と同じです。

論理配列

非複素数数値配列またはスパース配列は、論理配列としてフラグを付けることができます。論理配列に対するストレージは、非論理配列に対するストレージと同じです。

空配列

任意のタイプの MATLAB 配列は、空にすることができます。空の mxArray とは、少なくとも 1 つの次元が 0 と等しい mxArray です。たとえば、タイプ double の倍精度 mxArray は、m と n が 0 に等しく、pr が NULL であるとき空配列です。

データタイプの使用法

MATLAB の 6 個の基本データタイプは、double, char, sparse, uint8, cell, struct です。MATLAB がサポートする任意のデータタイプを受け取る MEX- ファイル、MAT- ファイルアプリケーション、エンジンアプリケーションを C で書くことができます。Fortran では、倍精度の n 行 m 列配列と文字列の作成のみがサポートされます。一度コンパイルすると C および Fortran MEX- ファイルは、M- ファンクションと同じように取り扱うことができます。

例題 explore

MATLAB に付属している、入力変数のデータタイプを識別する explore という例題の MEX- ファイルがあります。この例題のソースファイルは、<matlab>/extern/examples/mex ディレクトリにあります。ここで、<matlab> は MATLAB がシステム上でインストールされている最上位のディレクトリを表わします。たとえば、以下のようにタイプすると、

```
cd([matlabroot '/extern/examples/mex']);
x = 2;
explore(x);
```

以下の結果を生成します。

```
-------------------------------------------------
Name: x
Dimensions: 1x1
Class Name: double
-------------------------------------------------
    (1,1) = 2
```

explore は、任意のデータタイプを受け取ります。つぎの例を使って、explore を試してみてください。

```
explore([1 2 3 4 5])
explore 1 2 3 4 5
explore({1 2 3 4 5})
explore(int8([1 2 3 4 5]))
explore {1 2 3 4 5}
explore(sparse(eye(5)))
explore(struct('name', 'Joe Jones', 'ext', 7332))
explore(1, 2, 3, 4, 5)
```

MEX- ファイルの作成

本節では、つぎのトピックスを説明します。

- コンパイラの必要条件
- UNIX での設定のテスト
- Windows での設定のテスト
- オプションファイルの指定

コンパイラの必要条件

インストールした MATLAB には、API の操作を行うために必要なすべてのツールが含まれています。MATLAB は、LCC という PC 用の C コンパイラをもっていますが、Fortran コンパイラはもっていません。ユーザ自身の C コンパイラを利用することを選択する場合は、ANSI C コンパイラでなければなりません。また、Microsoft Windows プラットフォームで操作をする場合は、コンパイラは 32 ビット windows dynamically linked libraries (DLL) を作成できなければなりません。

MATLAB は、多くのコンパイラをサポートし、それらのコンパイラ用に特別に指定されたオプションファイルと呼ばれるファイルを提供します。「オプションファイル」の表は、サポートされているすべてのコンパイラと、それらの対応するオプションファイルの一覧です。多くのコンパイラのサポートの目的は、ユーザが選択するツールを使い易くするためです。しかし、多くの場合で、ユーザアプリケーションを作成するには、提供されている LCC コンパイラを単に使うだけで実現できます。

The MathWorks は、ウェブ： http://www.mathworks.com/support/tech-notes/v5/1600/1601.shtml において、MATLAB によってサポートされているコンパイラのリストを掲示しています。

注意　The MathWorks は、コンパイラの選択や切り替えを簡単に行うことができる mex スクリプト用のオプション (セットアップ) を提供しています。

つぎの節には、UNIX および Windows で MEX- ファイルを作成するための設定情報が含まれています。mex スクリプトに関するより詳しい情報は、「MEX- ファイルの作成のカスタマイズ」にあります。さらに、MEX- ファイルの作成において問題がある場合は、「トラブルシューティング」を参照してください。

UNIX での設定のテスト

システムが適切に MEX- ファイルを作成するように設定されているかどうかをチェックする最速の方法は、実際のプロセスを試してみることです。例として、<matlab>/extern/examples/mex ディレクトリに C のソースコード yprime.c とその Fortran 版 yprimef.F および yprimefg.F があります。ここで、<matlab> は、システム上で MATLAB がインストールされている最上位のディレクトリを表わします。

例題のソースファイル yprime.c または yprimef.F と yprimefg.F を UNIX 上でコンパイル、リンクするためには、まずそれらのファイルをローカルディレクトリにコピーし、それからローカルディレクトリにディレクトリの変更（cd）をしなければなりません。

MATLAB プロンプトで、つぎのようにタイプします。

```
mex yprime.c
```

これは、システムコンパイラを使って、システムに対応する拡張子をもつ yprime という MEX- ファイルを作成します。

M-function のように yprime を呼び出すことができます。

```
yprime(1,1:4)
ans =
    2.0000   8.9685   4.0000   -1.0947
```

例題プログラムの Fortran 版を Fortran コンパイラを使って行うためには、MATLAB プロンプトでつぎのようにタイプします。

```
mex yprimef.F yprimefg.F
```

MATLAB プロンプトから mex スクリプトを実行するのに加えて、システムプロンプトからもスクリプトを実行できます。

コンパイラの選択

デフォルトのコンパイラを変更するには、別のオプションファイルを選択します。つぎのコマンドを使って行うことができます。

```
mex -setup
```

```
 Using the 'mex -setup' command selects an options file that is
placed in ~/matlab and used by default for 'mex'. An options
file in the current working directory or specified on the
command line overrides the default options file in ~/matlab.
```

```
    Options files control which compiler to use, the compiler and
    link command options, and the runtime libraries to link
    against.

    To override the default options file, use the 'mex -f' command
    (see 'mex -help' for more information).

The options files available for mex are:

  1: <matlab>/bin/gccopts.sh :
       Template Options file for building gcc MEXfiles

  2: <matlab>/bin/mexopts.sh :
       Template Options file for building MEXfiles using the
        system ANSI compiler

Enter the number of the options file to use as your default options file:
```

番号を入力し、**Return** を押してシステムに対する適切なオプションファイルを選択します。オプションファイルが MATLAB ディレクトリにない場合は、システムはオプションファイルがユーザの matlab ディレクトリにコピーされたことを示すメッセージを表示します。オプションファイルが matlab ディレクトリに既に存在する場合は、システムは上書きするかどうかを尋ねます。

注意　　setup オプションは、ユーザのホームディレクトリにユーザ固有の matlab ディレクトリを作成し、適切なオプションファイルをそのディレクトリにコピーします (ディレクトリが既に存在する場合は、新たなファイルを作成しません)。この matlab ディレクトリは、ユーザのオプションファイルに対してのみ用いられます。ユーザは、独自のデフォルトオプションファイルをもつことができます (その他の MATLAB プロダクトが、このディレクトリにオプションファイルを置く場合があります)。これらのユーザ独自の matlab ディレクトリと、MATLAB がインストールされているシステムの matlab ディレクトリを混同しないでください。このディレクトリ名をマシン上で見るためには、MATLAB コマンド prefdir を使ってください。

---

setup オプションを使うと、デフォルトコンパイラをリセットするので、mex スクリプトを使うたびに新たなコンパイラが利用されます。

Windows での設定のテスト

Windows プラットフォームで MEX- ファイルを作成する前に、コンパイラに対するデフォルトのオプションファイル mexopts.bat を設定する必要があります。スイッチ setup は、デフォルトのオプションファイルを設定するための簡単な方法を提供します。オプションファイルを設定または変更するために、MATLAB または DOS コマンドプロンプトから

```
mex -setup
```

を実行します。

コンパイラの選択

mexスクリプトは、システム上にインストールされているCまたはC++コンパイラがなく、C MEX- ファイルをコンパイルしようとする場合には、LCC コンパイラを自動的に利用します。当然、Fortran プログラムをコンパイルする必要がある場合は、ユーザ独自の Fortran コンパイラを与えなければなりません。

mex スクリプトは、ファイル名の拡張子を使って、MEX- ファイルの作成に用いるコンパイラのタイプを決定します。たとえば、

```
mex test1.f
```

は、ユーザの Fortran コンパイラを利用し、

```
mex test2.c
```

は、ユーザの C コンパイラを利用します。

コンパイラがないシステム　システム上に C または C++ コンパイラがない場合は、mex ユーティリティは、インクルードされた LCC コンパイラを自動的に設定します。そのため、これらのシステム上で C MEX- ファイルを作成するためには、単につぎのように入力するだけでかまいません。

```
mex filename.c
```

MEX- ファイルの作成のこの簡単な方法は、ほとんどのユーザに対して機能します。

同梱の LCC コンパイラの利用がユーザのニーズを満たす場合は、1-17 ページの「MEX- ファイルの作成」をスキップしてかまいません。

コンパイラがあるシステム　C, C++，または Fortran コンパイラがシステム上にある場合は、利用したいコンパイラを選択できます。コンパイラを選択すると、そのコンパイラがデフォルトコンパイラになり、MEX- ファイルをコンパイルする

ときに選択する必要はありません。コンパイラを選択するか、既存のデフォルトコンパイラを変更するには、mex -setup を使います。

つぎの例題は、デフォルトコンパイラを Microsoft Visual C++ Version 6.0 コンパイラに設定するプロセスを示しています。

```
mex -setup

Please choose your compiler for building external interface (MEX) files.

Would you like mex to locate installed compilers [y]/n? n

Select a compiler:
[1] Borland C++Builder version 5.0
[2] Borland C++Builder version 4.0
[3] Borland C++Builder version 3.0
[4] Borland C/C++ version 5.02
[5] Borland C/C++ version 5.0
[6] Compaq Visual Fortran version 6.1
[7] Digital Visual Fortran version 5.0
[8] Lcc C version 2.4
[9] Microsoft Visual C/C++ version 6.0
[10] Microsoft Visual C/C++ version 5.0
[11] WATCOM C/C++ version 11
[12] WATCOM C/C++ version 10.6

[0] None

Compiler: 9

Your machine has a Microsoft Visual C/C++ compiler located at D:\Applications\Microsoft
Visual Studio. Do you want to use this compiler [y]/n? y

Please verify your choices:

Compiler: Microsoft Visual C/C++ 6.0
Location: D:\Applications\Microsoft Visual Studio

Are these correct?([y]/n): y

The default options file:
```

```
"C:\WINNT\Profiles\username\ApplicationData\MathWorks\MATLAB\R12\mexopts.bat"  is
being updated from ...
```

指定したコンパイラが存在しない場合は、つぎのメッセージが表示されます。

```
The default location for compiler-name is directory-name,
but that directory does not exist on this machine.

Use directory-name anyway [y]/n?
```

setup オプションを使うと、デフォルトコンパイラを設定するので、mex スクリプトを使うたびに新たなコンパイラが用いられます。

Windows での MEX- ファイルの作成

<matlab>\extern\examples\mex ディレクトリには、Cのソースコード yprime.c とその Fortran 版 yprimef.f および yprimefg.f があります。ここで、<matlab> は、システム上で MATLAB がインストールされている最上位のディレクトリを表わします。

Windows で例題のソースファイルをコンパイル、リンクするには、MATLAB プロンプトで、つぎのようにタイプします。

```
cd([matlabroot '\extern\examples\mex'])
mex yprime.c
```

Windowsプラットフォームに対応する.DLL拡張子をもつyprimeというMEX-ファイルを作成します。

M-function のように yprime を呼び出すことができます。

```
yprime(1,1:4)
ans =
    2.0000   8.9685   4.0000   -1.0947
```

サンプルプログラムのFortran版をFortranコンパイラで試してみるには、mex -setup を使って Fortran コンパイラに変更します。その後、MATLAB プロンプトでつぎのようにタイプします。

```
cd([matlabroot '\extern\examples\mex'])
mex yprimef.f yprimefg.f
```

MATLAB プロンプトから mex スクリプトを実行するのに加えて、システムプロンプトからスクリプトを実行することもできます。

オプションファイルの指定

UNIX および Windows で、-f オプションを使ってオプションファイルを指定することができます。-f オプションを利用するには、MATLAB プロンプトで

mex *filename* -f <*optionsfile*>

とタイプし、パス名と共にオプションファイル名を指定します。下記のオプションファイルの表には、MATLAB に付属するオプションファイルの一覧が含まれています。

mex スクリプトを使うたびにオプションファイルを指定する必要がある状況があります。つぎのような状況です。

- (*Windows および UNIX*) 他のコンパイラを使いたい (かつ -setup オプションを使わない)、あるいは MAT またはエンジンスタンドアロンプログラムをコンパイルしたい場合。
- *(UNIX)* システムの C コンパイラを使いたくない場合。

前もって設定されたオプションファイル

MATLAB には、特定のコンパイラで利用できる、前もって設定されたオプションファイルが付属しています。オプションファイルの表は、本リリースの MATLAB に含まれているオプションファイルをもつコンパイラの一覧です。

表 1-2: オプション ファイル

プラットフォーム	コンパイラ	オプションファイル
Windows	Borland C++, Version 5.0 & 5.2	bccopts.bat
	Borland C++Builder 3.0 (Borland C++, Version 5.3)	bcc53opts.bat
	Borland C++Builder 4.0 (Borland C++, Version 5.4)	bcc54opts.bat
	Borland C++Builder 5.0 (Borland C++, Version 5.5)	bcc55opts.bat
	Lcc C Compiler, MATLAB にバンドル	lccopts.bat

表 1-2: オプション ファイル

プラットフォーム	コンパイラ	オプションファイル
	Microsoft C/C++, Version 5.0	msvc50opts.bat
	Microsoft C/C++, Version 6.0	msvc60opts.bat
	Watcom C/C++, Version 10.6	watcopts.bat
	Watcom C/C++, Version 11	wat11copts.bat
	DIGITAL Visual Fortran, Version 5.0	df50opts.bat
	Compaq Visual Fortran, Version 6.1	df60opts.bat
	Borland C, Version 5.0 & 5.2, エンジンと MAT スタンドアロンプログラム用	bccengmatopts.bat
	Borland C, Version 5.3, エンジンと MAT スタンドアロンプログラム用	bcc53engmatopts.bat
	Borland C, Version 5.4, エンジンと MAT スタンドアロンプログラム用	bcc54engmatopts.bat
	Borland C, Version 5.5, エンジンと MAT スタンドアロンプログラム用	bcc55engmatopts.bat
	Lcc C compiler エンジンと MAT スタンドアロンプログラム用	lccengmatopts.bat
	Microsoft Visual C エンジンと MAT スタンドアロンプログラム用 Version 5.0	msvc50engmatopts.bat
	Microsoft Visual C エンジンと MAT スタンドアロンプログラム用 Version 6.0	msvc60engmatopts.bat

表 1-2: オプション ファイル

プラットフォーム	コンパイラ	オプションファイル
	Watcom C エンジンと MAT スタンドアロンプログラム用 Version 10.6	watengmatopts.bat
	Watcom C エンジンと MAT スタンドアロンプログラム用 Version 11	wat11engmatopts.bat
	DIGITAL Visual Fortran MAT スタンドアロンプログラム用 Version 5.0	df50engmatopts.bat
	Compaq Visual Fortran MAT スタンドアロンプログラム用 Version 6.1	df60engmatopts.bat
UNIX	System ANSI Compiler	mexopts.sh
	GCC	gccopts.sh
	System C++ Compiler	cxxopts.sh
	System ANSI Compiler エンジンおよび MAT スタンドアロンプログラム用	engopts.sh
	System ANSI Compiler MAT スタンドアロンプログラム用	matopts.sh

オプションファイルの最新のリストは、FTP サーバ、ftp://ftp.mathworks.com/pub/tech-support/docexamples/apiguide/R12/bin から入手可能です。MATLAB がサポートするすべてのコンパイラのリストは、http://www.mathworks.com/support の MathWorks Technical Support Web サイトにアクセスしてください。

注意　つぎの節「MEX- ファイルの作成のカスタマイズ」には、特定のシステムに対してオプションファイルを変更する方法に関する情報が含まれます。

MEX- ファイルの作成のカスタマイズ

本節では、MEX- ファイル作成スクリプトが利用するプロセスを詳しく説明します。つぎのトピックスが含まれます。

- この章の対象ユーザ
- MEX スクリプトスイッチ
- UNIX でのデフォルトオプションファイル
- Windows でのデフォルトオプションファイル
- UNIX での作成のカスタマイズ
- Windows での作成のカスタマイズ

この章の対象ユーザ

一般に、MATLAB のデフォルトで、ほとんどの MEX- ファイルの作成に対しては十分です。以下のような、より詳細な情報を必要とする理由があります。

- MEX- ファイルの作成のため提供されているスクリプトではなく Integrated Development Environment (IDE) を利用したい場合。
- たとえば、直接サポートされていないコンパイラを利用するため、新規のオプションファイルを作成したい場合。
- スクリプトの利用よりも作成プロセスに関する制御を行いたい場合。

一般にスクリプトは、MEX- ファイル作成のために 2 つのステージ (または Microsoft Windows では 3 つ) を使います。これらは、コンパイルとリンクです。これらの 2 つのステージの間に、Windows のコンパイラはリンクの準備 (プレリンクステージ) のためさらに処理を行わなければなりません。

MEX スクリプトスイッチ

mex スクリプトは、リンクとコンパイルステージを変更するためのスイッチ (オプションとも呼ばれます) をもっています。表「MEX スクリプトスイッチ」は、利用可能なスイッチとそれらの利用を示しています。各スイッチは、特に記述がない限り UNIX と Windows の両方で利用可能です。

ビルドプロセスのカスタマイズのため、オプションファイルを修正します。これは、システムが要求する通常のコンパイル、プレリンク、リンクの処理に対する

コンパイラ固有のフラグを含みます。オプションファイルは、変数の割り当てから成り立っています。変数は、作成プロセスの論理部分を表わします。

表 1-1: MEX スクリプトスイッチ

スイッチ	関数
@<rsp_file>	テキストファイル <rsp_file> の内容を mex スクリプトのコマンドライン引数として含みます。
-argcheck	MATLAB API関数 (Cの関数のみ) について引数チェックを行います。
-c	コンパイルのみ行い、リンクは行いません。
-D<name>[#<def>]	Cプリプロセッサマクロ<name> [値<def>もつ]を定義します (注意： UNIX も -D<name>[=<def>] を利用します)
-f <file>	<file>をオプションファイルとして使います。<file>は、カレントディレクトリにない場合は絶対パス名です。
-g	デバッグシンボルを含んだ実行ファイルをビルドします。
-h[elp]	ヘルプ。スイッチとそれらの機能を表示します。
-I<pathname>	コンパイラの include サーチパスに <pathname> をインクルードします。
-inline	行列アクセス関数 (mx*) をインライン化します。生成された MEX-function は、MATLAB の将来のバージョンで適用可能でない場合があります。
-l<file>	(UNIX) ライブラリ lib<file> をリンクします。
-L<pathname>	(UNIX) ライブラリの検索のためのディレクトリリストに <pathname> をインクルードします。

表 1-1: MEX スクリプトスイッチ

スイッチ	関数
<name>#<def>	変数 <name> に対するオプションファイルの設定を変更します。このオプションは、mex の呼び出し中に環境変数 <ENV_VAR> を一時的に <val> に設定する <ENV_VAR>#<val> と等価です。<val> は、変数名に $ を付けることによって、他の環境変数を参照することが可能です。例 COMPFLAGS#"$COMPFLAGS -myswitch"
<name>=<def>	(UNIX) 変数 <name> に対するオプションファイルの設定を変更します。
-O	最適化された実行ファイルをビルドします。
-outdir <name>	すべての出力ファイルをディレクトリ <name> に置きます。
-output <name>	<name> という実行ファイルを作成します (適切な実行可能な拡張子が自動的に加えられます)。
-setup	デフォルトのオプションファイルを設定します。このスイッチは、他の引数を組み合わせません。
-U<name>	C プリプロセッサマクロ <name> を未定義にします。
-v	冗長。すべてのコンパイラとリンカの設定をプリントします。
-V4	MATLAB 4 互換の MEX- ファイルをコンパイルします。

UNIX でのデフォルトオプションファイル

MATLAB に付属するデフォルトの MEX オプションファイルは、<matlab>/bin にあります。mex スクリプトは、つぎの順番にオプションファイル mexopts.sh を検索します。

- カレントディレクトリ
- $HOME/matlab
- <matlab>/bin

mex は、最初に見つかったオプションファイルを使います。オプションファイルが見つからない場合は、mex はエラーメッセージを表示します。-f スイッチを使ってオプションファイル名を直接指定できます。

MATLAB がサポートするコンパイラに関するデフォルトの設定についての情報は、<matlab>/bin/mexopts.sh のオプションファイルを調べたり、verbose モード (-v) で mex スクリプトを呼び込むことができます。Verbose モードは、各コンパイラに対するビルドプロセスで用いられる正確なコンパイルのオプション、プレリンクコマンド、リンカオプションを表示します。「UNIX での作成のカスタマイズ」には、高レベルのビルドプロセスの概要があります。

Windows でのデフォルトオプションファイル

デフォルトの MEX オプションファイルは、mex -setup を実行してシステムを設定した後で、ユーザのプロファイルディレクトリに置かれます。mex スクリプトは、つぎの順番で mexopts.bat と呼ばれるオプションファイルをサーチします。

- カレントディレクトリ
- ユーザのプロファイルディレクトリ。このディレクトリに関する詳細は、「ユーザプロファイルディレクトリ」を参照してください。
- <matlab>\bin\win32\mexopts

mex は、最初に見つかったオプションファイルを使います。オプションファイルが見つからない場合は、mex はマシン上のサポートされている C コンパイラをサーチし、そのコンパイラを利用するように自動的に設定します。また、設定処理中に、コンパイラのデフォルトオプションファイルをユーザプロファイルディレクトリにコピーします。複数のコンパイラが見つかった場合は、1 つを選択するように表示されます。

MATLAB がサポートするコンパイラのデフォルトの設定に関する固有の情報は、オプションファイル mexopts.bat を調べるか、verbose モード (-v) で mex スクリプトを呼び込むことができます。Verbose モードは、各コンパイラに対するビルドプロセスで用いられる正確なコンパイラオプション、プレリンクコマンド (適切である場合)、リンカオプションを表示します。「Windows での作成のカスタマイズ」には高レベルのビルドプロセスの概要が説明されています。

ユーザプロファイルディレクトリ

Windows のユーザのプロファイルディレクトリは、デスクトップの外観、最近使ったファイル、**スタート**メニューアイテムのような、ユーザ固有の情報を含むディレクトリです。mex および mbuild ユーティリティは、オプションファイル mexopts.bat と compopts.bat を格納しています。これらのファイルは、Application

Data\MathWorks\MATLAB というユーザプロファイルディレクトリのサブディレクトリに -setup プロセス中に作成されます。ユーザプロファイルが利用可能な Windows NT および Windows 95/98 では、ユーザプロファイルディレクトリは、%windir%\Profiles\username です。ユーザプロファイルが利用不可能な Windows 95/98 では、ユーザプロファイルディレクトリ %windir% です。Windows 95/98 では、**Password** コントロールパネルを使うことによって、ユーザプロファイルが利用可能かどうかを決定することができます。

UNIX での作成のカスタマイズ

UNIX システムでは、MEX- ファイルの作成には、コンパイルとリンクの 2 つのステージがあります。

コンパイルステージ

コンパイルステージでは、以下のことを行う必要があります。

- ヘッダファイルを検索するディレクトリのリスト (-I<matlab>/extern/include) に <matlab>/extern/include を追加します。
- プリプロセッサマクロ MATLAB_MEX_FILE (-DMATLAB_MEX_FILE) を定義します。
- (C MEX- ファイルのみ)MEX- ファイルのバージョン情報を含むソースファイル <matlab>/extern/src/mexversion.c をコンパイルします。

リンクステージ

リンクステージでは、以下のことを行う必要があります。

- 共有ライブラリを構築するためにリンカに指示します。
- コンパイルされたソースファイル (mexversion.c を含む) からすべてのオブジェクトをリンクします。
- (Fortran MEX- ファイルのみ) プレコンパイルされたバージョンのソースファイル <matlab>/extern/lib/$Arch/version4.o をリンクします。
- シンボル mexFunction と mexVersion エクスポートします (これらのシンボルは、MATLAB に呼び出される関数を表わします)。

Fortran MEX- ファイルに対して、シンボルはすべて小文字で、アンダースコアが加えられることがあります。個々の情報に対しては、verbose モードで mex スクリプトを呼び込み、出力を調べてください。

ビルドオプション

作成プロセスのカスタマイズのために、オプションファイルを修正する必要があります。オプションファイルは、上記の一般の処理に対応するコンパイラ固有のフラグを含みます。オプションファイルは、変数の割り当てから成り立っています。各変数は、作成プロセスの論理部分を表わします。MATLAB が提供するオプションファイルは、<matlab>/bin にあります。mex スクリプトがオプションファイルを検索する方法は、「UNIX でのデフォルトオプションファイルの位置」に記述されています。

カスタマイズを柔軟に行うために、オプションファイル内でスイッチを使って mex スクリプトの on と off を行う 2 セットのオプションがあります。これらのオプションは、" *デバッグモード* " での作成と " *最適化モード* " での作成に対応します。これらはそれぞれ、呼び込まれる " ドライバ " に対して 1 組の変数 DEBUGFLAGS と OPTIMFLAGS によって表わされます (C コンパイラに対しては CDEBUGFLAGS、Fortran コンパイラに対しては FDEBUGFLAGS、リンカに対しては LDDEBUGFLAGS で、OPTIMFLAGS も同様です)。

- 最適化モードで作成する場合 (デフォルト)、mex スクリプトはコンパイルおよびリンクステージで OPTIMFLAGS オプションをインクルードします。
- デバッグモードで作成する場合、mex スクリプトはコンパイルおよびリンクステージで DEBUGFLAGS オプションをインクルードしますが、OPTIMFLAGS オプションはインクルードしません。
- mex スクリプトに最適化とデバッグのフラグ (それぞれ -O と -g) の両方を指定することで、両方のオプションをインクルードすることができます。

これらの特殊な変数とは別に、mex オプションファイルは、3 つのモード (C のコンパイル、Fortran のコンパイル、リンク) に対し呼び込まれる実行ファイルと、各ステージに対するフラグを定義します。各言語のソースファイルを含む MEX- ファイルに対してリンクされなければならないライブラリのリストを与えることもできます。

変数は、つぎのようにまとめられます。

変数	C コンパイラ	Fortran コンパイラ	リンカ
実行可能	CC	FC	LD
フラグ	CFLAGS	FFLAGS	LDFLAGS
最適化	COPTIMFLAGS	FOPTIMFLAGS	LDOPTIMFLAGS

変数	C コンパイラ	Fortran コンパイラ	リンカ
デバッグ	CDEBUGFLAGS	FDEBUGFLAGS	LDDEBUGFLAGS
追加ライブラリ	CLIBS	FLIBS	----

これらの変数のデフォルト設定の特性について、つぎのことを行えます。

- <matlab>/bin/mexopts.shにあるオプションファイル (または使っているオプションファイル) を調べます。
- verbose モードで mex スクリプトを呼び込みます。

Windows での作成のカスタマイズ

Windows での C と Fortran の MEX- ファイルの作成には、コンパイル、プレリンク、リンクの 3 つのステージがあります。

コンパイルステージ

コンパイルステージでは、mex オプションファイルは、つぎのことを行う必要があります。

- COMPILER (例， Watcom), PATH, INCLUDE, LIB 環境変数を使って、コンパイラのパスを設定します。コンパイラが常に環境変数のセット (例，AUTOEXEC.-BAT) をもっている場合は、オプションファイル内でそれらのマークを外すことができます。
- 必要ならば、COMPILER 環境変数を使ってコンパイラ名を定義します。
- COMPFLAGS 環境変数で、コンパイラのスイッチを定義します。
  - **a** MEX- ファイルに対しては、DLL を作成するスイッチが必要です。
  - **b** スタンドアロンプログラムに対しては、exeを作成するスイッチが必要です。
  - **c** -c スイッチ (コンパイルのみで、リンクは行いません) を推奨します。
  - **d** 8 バイトの配置を指定するスイッチ。
  - **e** 環境を指定する他のスイッチを使うことができます。
- -D, MATLAB_MEX_FILE によるプリプロセッサ macro の定義が必要です。
- OPTIMFLAGSとDEBUGFLAGSを使って最適化スイッチとデバッグスイッチを設定します。これらは排他的です。OPTIMFLAGS がデフォルトで、mex のコマンドラインで -g を指定すれば、DEBUGFLAGS が使われます。

プレリンクステージ

プレリンクステージは、MEX, MAT, エンジンファイルに要求された関数を読み込むためのインポートライブラリをダイナミックに作成します。すべての MEX- ファイルは、MATLAB のみをリンクします。MAT スタンドアロンプログラムは、libmx.dll (配列のアクセスライブラリ) と libmat.dll（MAT- ファンクション) をリンクします。エンジンスタンドアロンプログラムは、libmx.dll 配列のアクセスライブラリ ）とエンジンファンクションに対して、libeng.dll をリンクします。MATLAB と DLL は、<matlab>\extern\include ディレクトリにある同じ名前の対応する .def ファイルをもちます。

リンクステージ

最後に、リンクステージでは、mex オプションファイルは、以下のことを行う必要があります。

- LINKER 環境変数でリンカ名を定義します。
- つぎのものを含む LINKFLAGS 環境変数を定義しなければなりません。
  - MEX- ファイルに対する DLL を作成するスイッチ、または、スタンドアロンプログラムに対する exe を作成するスイッチ。
  - C に対して、 mexFunction、DIGITAL Visual Fortran に対して、MEXFUNCTION@16 のように、MEX- ファイルへのエントリポイントをエクスポートします。
  - PRELINK_CMDS ステージで作成したインポートライブラリ。
  - 使用するコンパイラ固有のリンクスイッチ。
- LINKEROPTIMFLAGS と LINKDEBUGFLAGS で、リンクの最適化スイッチとデバッグスイッチを定義します。コンパイルステージのように、これらは排他的です。デフォルトは最適化で、-g スイッチがデバッグスイッチを呼び込みます。
- 必要ならば、LINK_FILE 環境変数でリンクファイル識別子を定義します。たとえば、Watcom は、file の後に続く名前がリンクファイル名であり、コマンドでないことを識別します。
- 必要ならば、LINK_LIB 環境変数でリンクライブラリ識別子を定義します。たとえば、Watcom は library の後に続く名前がライブラリ名であり、コマンドでないことを識別します。
- オプションで、NAME_OUTPUT 環境変数で出力スイッチを使って識別子とファイル名を設定します。環境変数 MEX_NAME は、コマンドラインの最初のプログラム名を含みます。これを動作させるには、-output に設定する必要があります。この環境変数が設定されていない場合は、コンパイラのデフォルトは、コ

マンドラインの最初のプログラム名を使います。設定されていても、mex -output スイッチを設定することで書き換えることができます。

MEX- ファイルへの DLL のリンク

DLL を MEX- ファイルにリンクするには、コマンドラインで DLL の .lib ファイルをリストします。

MEX- ファイルのバージョン設定

mex スクリプトは、バージョン設定やその他の重要な情報を含むリソースファイルと共にユーザの MEX- ファイルを作成します。リソースファイルは、mexversion.rc という名前で、extern\include ディレクトリにあります。バージョン設定をサポートするために、リソースコンパイラとリンカコマンドを与える 2 つの新しいコマンド RC_COMPILER と RC_LINKER がオプションファイルにあります。つぎのように考えられます。

- コンパイラコマンドが与えられると、標準のリンクコマンドを使って、コンパイルされたリソースが MEX- ファイルにリンクされます。
- リンカコマンドが与えられると、リソースファイルは、そのコマンドを使って作成された MEX- ファイルにリンクされます。

Microsoft Visual C++ IDE を使った MEX- ファイルのコンパイル

注意　この節では、Microsoft Visual C++ (MSVC) IDE(これは、含まれていません）で MEX- ファイルをコンパイルする方法を説明しています。この節では、ユーザは IDE の利用法を理解していると仮定しています。MSVC IDE の利用に関する詳しい情報は、Microsoft のドキュメントを参照してください。

Microsoft Visual C++ 統合開発環境を使って MEX- ファイルを作成するためには、つぎのようにします。

1 プロジェクトを作成し、そこに MEX のソースと mexversion.rc を挿入します。

2 MEX のエントリポイントをエクスポートするために、.DEF ファイルを作成します。たとえば、

```
LIBRARY MYFILE.DLL
EXPORTS mexFunction             <-- C MEX- ファイル
```

または

```
EXPORTS MEXFUNCTION@16          <-- Fortran MEX- ファイル
```

3 プロジェクトに .DEF ファイルを追加します。

4 利用するコンパイラのバージョンに対応する .LIB ファイルは、matlabroot\extern\lib\win32\microsoft にあります。たとえば、version 6.0 に対しては、これらのファイルは、msvc60 サブディレクトリにあります。

5 このディレクトリから、LINK 設定オプションで libmx.lib, libmex.lib, libmat.lib をライブラリモジュールに追加します。

6 MATLABのincludeディレクトリMATLAB\EXTERN\INCLUDEを**Settings C/C++ Preprocessor** オプションの include パスに追加します。

7 ビルドメニューから**Settings**を選択し、**C/C++**を選択し、**Preprocessor definitions** フィールドの最後の要素の後にMATLAB_MEX_FILEをタイプすることにより MATLAB_MEX_FILE を **C/C++ Preprocessor** オプションに追加します。

8 IDE を使って、MEX- ファイルをデバッグするには、**MATLAB.EXE を Settings Debug** オプションに **Executable for debug session** としておきます。

Microsoft Visual C/C++ コンパイラ以外のコンパイラを利用している場合は、MEX- ファイルのビルドプロセスは、上述したものと同じです。ステップ 4 では、利用するコンパイラ用の .LIB ファイルは matlabroot\extern\lib\win32 のサブディレクトリにあります。たとえば、Borland C/C++ compiler の version 5.4 については、matlabroot\extern\lib\win32\borland\bc54 を見てください。

Visual Studio 用アドインの使用法

The MathWorks は、Microsoft Visual C/C++ (MSVC) 内で簡単に利用できる Visual Studio® 開発システム用の MATLAB アドインを提供します。Visual Studio 用 MATLAB アドインは、MSVC 環境での M- ファイルの利用を大きく簡略化します。アドインは、M- ファイルと Visual C++ プロジェクトとの統合を自動化します。MSVC 環境と完全に統合されています。

Visual Studio 用アドインは、mbuild -setup あるいは mex -setup を実行し、Microsoft Visual C/C++ version 5 または 6 を選択したときに自動的にシステムにインストールされます。しかし、アドインを利用するために以下のステップを行う必要があります。

1 Visual Studio 用アドインを使って MEX- ファイルを作成するには、MATLAB コマンドプロンプトで以下のコマンドを実行してください。

```
mex -setup
```

メニューに従い、Microsoft Visual C/C++ 5.0 または 6.0 を選択します。これにより、選択された Microsoft compiler を利用する mex を作成し、Microsoft Visual C/C++ ディレクトリに必要なアドインファイルをインストールします。

2 Microsoft Visual C/C++ と動作する Visual Studio 用 MATLAB アドインを設定するには、以下のようにします。

   a MSVC メニューから **Tools -> Customize** を選択します。

   b **Add-ins and Macro Files** タブをクリックします。

   c **Browse** をクリックし、ファイル名として <matlab>\bin\win32 を入力し、**Files of Type** プルダウンリストから Add-ins (.dll) を選択します。

注意　Windows が " システムファイルを表示しない " に設定されている場合は、.dll ファイルは、ディスク上に存在しても表示されません。この機能を利用不可能にするには、表示オプションを変更する必要があります。

---

   d MATLABAddin.dll ファイルを選択し、**Open** をクリックします。

   e **Add-ins and Macro Files** リストの **MATLAB for Visual Studio** をチェックし、**Close** をクリックします。Visual Studio ツールバーに MATLAB アドインが現れます。チェックマークは、MSVC の起動時にアドインを自動的にロードすることを示します。

注意　Windows 95 あるいは Windows 98 システムで Visual Studio 用 MATLAB アドインを実行するには、config.sys ファイルに以下の行を付け加えます。

```
shell=c:\command.com /e:32768 /p
```

Visual Studio 用 MATLAB アドインに関する情報は、以下を参照してください。

- <matlab>\bin\win32 ディレクトリの MATLABAddin.hlp ファイルを参照してください。あるいは、
- Visual Studio ツールバーの MATLAB アドインのヘルプアイコンをクリックしてください。

トラブルシューティング

本節では、発生する一般的な問題に対するトラブルシュートの方法を説明します。つぎのトピックスを含みます。

- 設定の問題
- MEX- ファイルの問題の理解
- コンパイラとプラットフォーム固有の問題
- メモリ管理の互換性の問題

設定の問題

この節では、MEX- ファイルの作成時に発生する共通の問題について説明します。

Windows でのサーチパスの問題

Windowsでは、MATLABを再インストールしないでMATLAB実行ファイルを移動すると、mex.bat を変更して新規の MATLAB の位置をポイントさせる必要があります。

Windows での空白を含む MATLAB のパス名

Windows において MEX- ファイルのビルドに問題があり、MATLAB パス内のディレクトリ名に空白がある場合は、空白を含まないパス名に MATLAB を再インストールするか、あるいは空白を含むディレクトリ名を変更する必要があります。たとえば、Program Files ディレクトリに MATLAB をインストールする場合、C コンパイラの中には、MEX- ファイルのビルドに問題が起こる場合があります。また、MATLAB V5.2 のようなディレクトリに MATLAB をインストールする場合にも、問題が起こる場合があります。

Windows において DLL がパスにない

MATLAB は、MEX- ファイルによって参照されるすべての DLL が見つからない場合は、MEX- ファイルのロードができません。DLL は、DOS のパス上か、あるいは、MEX- ファイルと同じディレクトリになければなりません。これは、サードパーティの DLL についても同様です。

一般的な設定の問題

この章に記述されているプラットフォーム用の設定ステップに従っていることを確認してください。また、詳細は、1-20 ページの「MEX- ファイルの作成のカスタマイズ」を参照してください。

MEX- ファイルの問題の理解

この節では、MEX- ファイルの作成時に発生する一般的な問題に関する情報を提供します。下記の図を使って、これらの問題を別々に考えます。

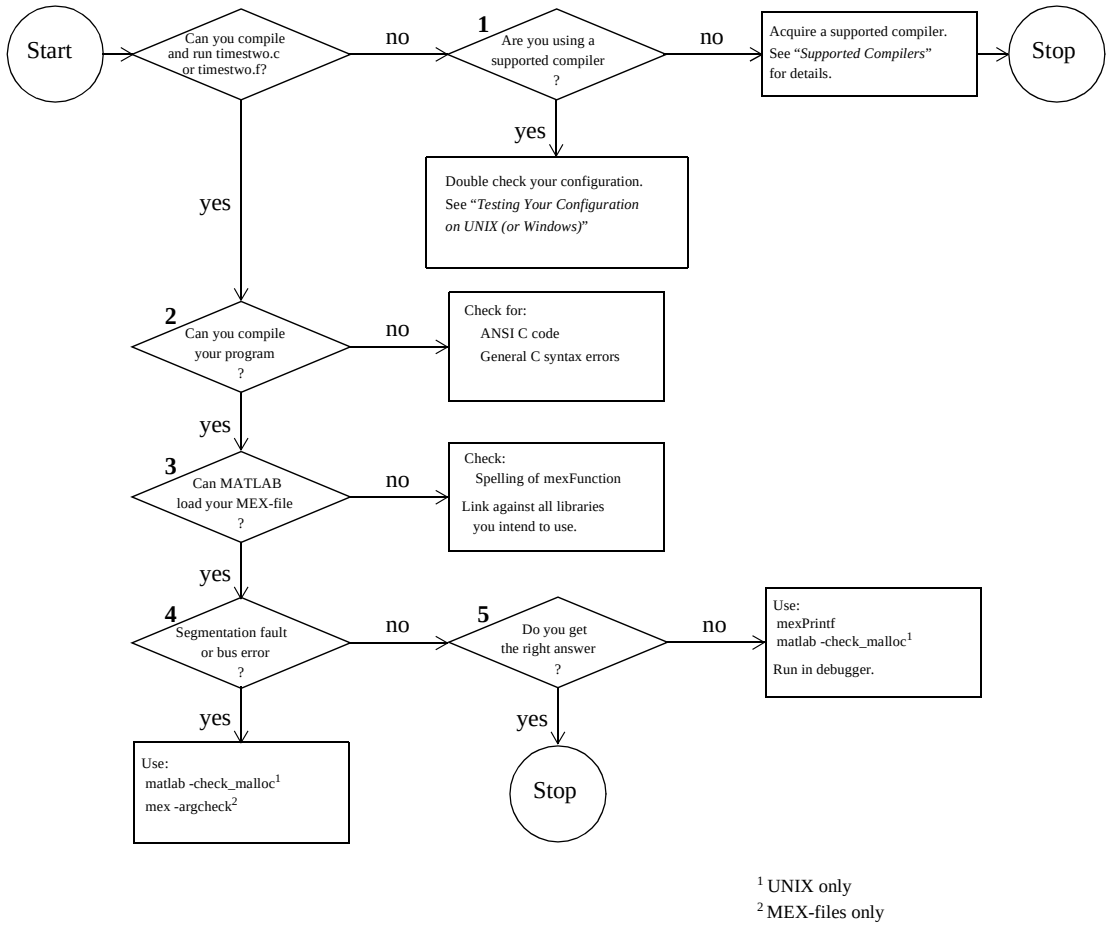


図 1-1: MEX- ファイルの作成の問題のトラブルシューティング

問題 1 から 5 は、上のフローチャートの特定の場所を参照します。MEX のビルドの問題を解決するためのヒントは、MathWorks Technical Support Web サイト、http://www.mathworks.com/support にアクセスしてください。

問題 1 - MathWorks プログラムのコンパイルの失敗

UNIX での C MEX-ファイルの作成の最も一般的な設定の問題は、非 ANSI C コンパイラの使用や、ANSI C コードをコンパイルすることを知らせるためのフラグをコンパイラに渡すことができないことです。

このタイプの設定の問題があるかどうかを知るための確かな方法は、The MathWorks により提供されるヘッダファイルが、コードをコンパイルしようとするときにシンタックスエラー文字列を生成するかどうかです。適切なオプションファイルを選択するための情報、または必要ならば ANSI C コンパイラを取得するための情報は「MEX- ファイルの作成」を参照してください。

問題 2 - ユーザプログラムのコンパイルの失敗

シンタックスエラー文字列を出力する第二の場合は、ANSI C コードと非 ANSI C コードが混在するときに起こります。The MathWorks は、ANSI C 準拠のヘッダファイルとソースファイルを提供しています。従って、ユーザの C コードも ANSI 準拠でなければなりません。

C プログラムで起こるその他の一般的な問題は、必要なすべてのヘッダファイルをインクルードするのを無視することや、必要なすべてのライブラリをリンクするのを無視することです。

問題 3 - MEX- ファイルのロードエラー

つぎの形式のエラー

```
Unable to load mex file:
??? Invalid MEX-file
```

が表示された場合、MATLAB はユーザの MEX- ファイルが有効であると認識できません。

MATLAB は、ゲートウェイルーチン mexFunction を探すことで MEX- ファイルをロードします。関数名のスペルを間違えた場合は、MATLAB は MEX- ファイルをロードできず、エラーメッセージを表示します。Windows では、正確に mexFunction を読み込んでいることをチェックしてください。

プラットフォームの中には、要求されたライブラリをリンクできない場合、MEX-ファイルをコンパイルするときでなく、MEX- ファイルをロードするときにエ

ラーが発生することがあります。そのような場合、"*unresolved symbols*" または "*unresolved references*" というシステムエラーメッセージが表示されます。関数を定義するライブラリをリンクしていることを確かめてください。

Windows では、MATLAB は MEX- ファイルで参照されるすべての DLL を見つけることができない場合、MEX- ファイルをロードできません。DLL は、パス上または MEX- ファイルと同じディレクトリになければなりません。これは、サードパーティの DLL に対しても同様です。

問題 4 - Segmentation Fault、または、Bus Error

MEX- ファイルが segmentation violation、または、バスエラーを起こす場合、プロテクトされていて、リードオンリーか、または、割り当てをされていないメモリに、MEX- ファイルがアクセスしようとしたことを意味します。これは一般的なプログラミングエラーなので、このような問題はしばしば解決が困難です。

Segmentation violations は、常にそれらの原因となる論理エラーと同じ点で起こるわけではありません。プログラムがメモリの意図しない部分にデータを書き出す場合、プログラムがデータを読み込んで解釈するまでエラーは起こりません。従って、segmentation violation またはバスエラーは、MEX- ファイルが実行を終了した後に起こります。

MATLAB は、この現象のトラブルシューティングの手助けとして、3 つの機能を提供します。以下に簡単な順に説明します。

- **引数チェックを行って MEX- ファイルを再コンパイルしてください (C MEX- ファイルのみ)。** mex スクリプトのフラグ -argcheck を使って再コンパイルすることで、MEX- ファイルにエラーチェックのレイヤーを追加できます。これは、MATLAB MEX- ファイル (mex) と行列のアクセス (mx) の API 関数の両方に無効な引数について警告します。

  この場合、MEX- ファイルは効率的に実行されませんが、このスイッチにより API 関数へ null ポインタを渡すようなエラーを検出します。

- **-check_malloc オプションを使って MATLAB を実行します (UNIX のみ)。** MATLAB のスタートアップフラグ -check_malloc は、MATLAB がメモリのチェックの情報を保持することを意味します。メモリが開放されるとき、MATLAB はこのメモリの前後のメモリが書き込まれていないことと、メモリが以前に開放されていないことを確かめます。

  エラーが起こる場合、MATLAB は割り当てられたメモリブロックのサイズを記録します。この情報を使って、このメモリの割り当てられたコードを追跡して、続けることができます。

このフラグを使うと MATLAB は効率的に実行することができませんが、配列の末尾の後に書き込んだり、以前に開放したメモリを開放するエラーを検出します。

- **デバッグ環境で MATLAB を実行します。**このプロセスは、C および Fortran MEX-ファイルの作成の章で既に説明しています。

問題 5 - プログラムが間違った結果を生成する

プログラムが間違った答えを出力する場合、いくつかの原因があります。第一に、計算ロジックでのエラーがあります。第二に、プログラムがメモリの初期化されていない部分から読み込まれている可能性があります。たとえば、10 要素のベクトルの 11 要素目を読み込むと、予期しない結果になります。

その他の可能性は、メモリの間違ったハンドリングによる有効なデータの上書きによるものです。たとえば、10 要素のベクトルの 15 要素目に書き込むと、メモリ内の隣接する変数のデータを上書きします。この場合、問題 4 で説明した segmentation violations と同様の方法で扱われます。

これらのすべての場合で、mexPrintf を使って中間のステージでデータ値を調べたり、デバッガが提供するすべてのツールを利用してデバッガ内で MATLAB を実行することができます。

コンパイラとプラットフォーム固有の問題

この節では、特定のコンパイラとプラットフォームに固有の状況を説明します。

Watcom IDE で作成された MEX- ファイル

Watcom IDE を使って MEX- ファイルを作成し、ライブラリのリンク時に API 関数への参照エラーが発生する場合は、引数の用法をチェックしてください。Watcom IDE は、パラメータをレジスタに渡すデフォルトのスイッチを利用します。MATLAB は、パラメータをスタックに渡すことを要求します。

メモリ管理互換性の問題

パフォーマンス問題に注目すると、MATLAB 内部のメモリ管理モデルに変更を行う必要があります。これらの変更は、MEX- ファイル API の将来の機能強化の準備となっています。

MATLAB 5.2 のように、MATLAB は左辺のリスト (plhs[]) に出力されない任意の mxArray において MEX- ファイルの実行の最後に mxArray のデストラクタ mxDestroyArray を暗示的に呼び出します。MATLAB が誤って作成されたあるいは不適切に破棄された mxArray を検出した場合、ワーニングを発生します。

つぎの節で説明するワーニングを生成する MEX- ファイルのコードを修正することを強く推奨します。詳細は、「C 言語 MEX- ファイルの作成」の「メモリ管理」を参照してください。

注意　現時点では、下記のワーニングは、下位互換性の理由によりデフォルトで利用可能です。将来の MATLAB のリリースでは、このワーニングは、デフォルトで利用不可能になる予定です。プログラマは、MEX- ファイルの開発サイクルにおいて、これらのワーニングを利用可能にする必要があります。

不適切に mxArray を破棄する

mxFree を使って、mxArray を破棄することはできません。

ワーニング

Warning: You are attempting to call mxFree on a <class-id> array. The destructor for mxArrays is mxDestroyArray; please call this instead. MATLAB will attempt to fix the problem and continue, but this will result in memory faults in future releases.

ワーニングが発生する例

つぎの例題では、mxFree は配列オブジェクトを破棄しません。このオペレーションは、配列に関連する構造体のヘッダを開放しますが、MATLAB は配列オブジェクトを破棄する必要があるかのように機能します。そのため、MATLAB は配列オブジェクトを破棄しようとし、さらにそのプロセスで再度構造体のヘッダを開放しようとします。

```
mxArray *temp = mxCreateDoubleMatrix(1,1,mxREAL);
   ...
mxFree(temp);  /* INCORRECT */
```

解決法

代わりに mxDestroyArray を呼び出します。

```
mxDestroyArray(temp);  /* CORRECT */
```

セル、または、構造体の mxArray を不適切に作成

mxSetCell、または、mxSetField を prhs[] によってメンバ配列として呼び出すことはできません。

ワーニング

Warning: You are attempting to use an array from another scope (most likely an input argument) as a member of a cell array or structure. You need to make a copy of the array first. MATLAB will attempt to fix the problem and continue, but this will result in memory faults in future releases.

ワーニングが発生する例

つぎの例で、MEX- ファイルがリターンするとき、MATLAB はセル配列全体を破棄します。これは、セルのメンバを含むので、MEX- ファイルの入力引数を暗示的に破棄します。これにより、右辺引数がテンポラリ配列である（ リテラルの、または式の結果) 場合、通常呼び出し側のワークスペースの破棄に関連する奇妙な結果が生じる場合があります。

```
myfunction('hello')
/* myfunction is the name of your MEX-file and your code */
/* contains the following:    */

  mxArray *temp = mxCreateCellMatrix(1,1);
   ...
  mxSetCell(temp, 0, prhs[0]);  /* INCORRECT */
```

解決法

mxDuplicateArrayを使って右辺引数のコピーを作り、そのコピーをmxSetCell (または mxSetField の引数として使います。たとえば、

```
mxSetCell(temp, 0, mxDuplicateArray(prhs[0]));  /* CORRECT */
```

テンポラリな mxArray を不適切なデータで作成

API ルーチンによって割り当てられていないデータをもつ mxArray について mxDestroyArray を呼び出すことはできません。

ワーニング

Warning: You have attempted to point the data of an array to a block of memory not allocated through the MATLAB API. MATLAB will attempt to fix the problem and continue, but this will result in memory faults in future releases.

ワーニングが発生する例

mxSetPr, mxSetPi, mxSetData, mxSetImagData を呼び出し、mxCalloc, mxMalloc, mxRealloc によって割り当てられていないメモリをデータブロック (第二引数) として指定する場合、MEX- ファイルがリターンするとき、MATLAB は実数データのポインタと虚数データ (存在すれば) のポインタを開放しようとします。そのため、MATLAB は、この例ではプログラムスタックからメモリを開放しようとします。これにより、MATLAB が互換性のチェックの情報を一致させようとするときに、上記のワーニングを発生します。

```
mxArray *temp = mxCreateDoubleMatrix(0,0,mxREAL);
  double data[5] = {1,2,3,4,5};
    ...
  mxSetM(temp,1); mxSetN(temp,5); mxSetPr(temp, data);
  /* INCORRECT */
```

解決法

データポインタを設定するために mxSetPr を使うよりも、正しいサイズで mxArray を作成し、memcpy を使ってスタックデータを mxGetPr によって出力されるバッファにコピーします。

```
mxArray *temp = mxCreateDoubleMatrix(1,5,mxREAL);
double data[5] = {1,2,3,4,5};
  ...
memcpy(mxGetPr(temp), data, 5*sizeof(double));  /* CORRECT */
```

潜在的なメモリリーク

バージョン 5.2 以前は、API 作成ルーチンを使って mxArray を作成し、mxSetPr を使ってデータのポインタを上書きする場合には、MATLAB は、オリジナルのメモリを開放していました。これは現在は行われません。

たとえば、

```
pr = mxCalloc(5*5, sizeof(double));
... <load data into pr>
plhs[0] = mxCreateDoubleMatrix(5,5,mxREAL);
mxSetPr(plhs[0], pr);  /* INCORRECT */
```

は、メモリが 5*5*8 バイトリークします。ここで、8 バイトは double のサイズです。

コードを変更してメモリリークを防ぐことができます。

```
plhs[0] = mxCreateDoubleMatrix(5,5,mxREAL);
pr = mxGetPr(plhs[0]);
... <load data into pr>
```

あるいは、つぎのようにします。

```
pr = mxCalloc(5*5, sizeof(double));
... <load data into pr>
plhs[0] = mxCreateDoubleMatrix(5,5,mxREAL);
mxFree(mxGetPr(plhs[0]));
mxSetPr(plhs[0], pr);
```

最初の解決法のほうが効率的です。

同様のメモリリークは、mxSetPi, mxSetData, mxSetImagData, mxSetIr, mxSetJc を使ったときにも起こる場合があります。メモリリークを防ぐために、上記のようにこの問題に対応することができます。

MEX- ファイルは、自身のテンポラリ配列を破棄します

一般に、MEX- ファイルがテンポラリ配列を破棄し、テンポラリメモリをクリーンアップすることを推奨します。左辺のリストに出力されるものと mexGetArrayPtr によって出力するもの以外のすべての mxArray は、安全に破棄されます。この方法は、他の MATLAB API アプリケーションとの整合性があります (例 , 自動クリーンアップメカニズムをもたない MAT- ファイルアプリケーション、エンジンアプリケーション、MATLAB Compiler 生成アプリケーション)。

その他の情報

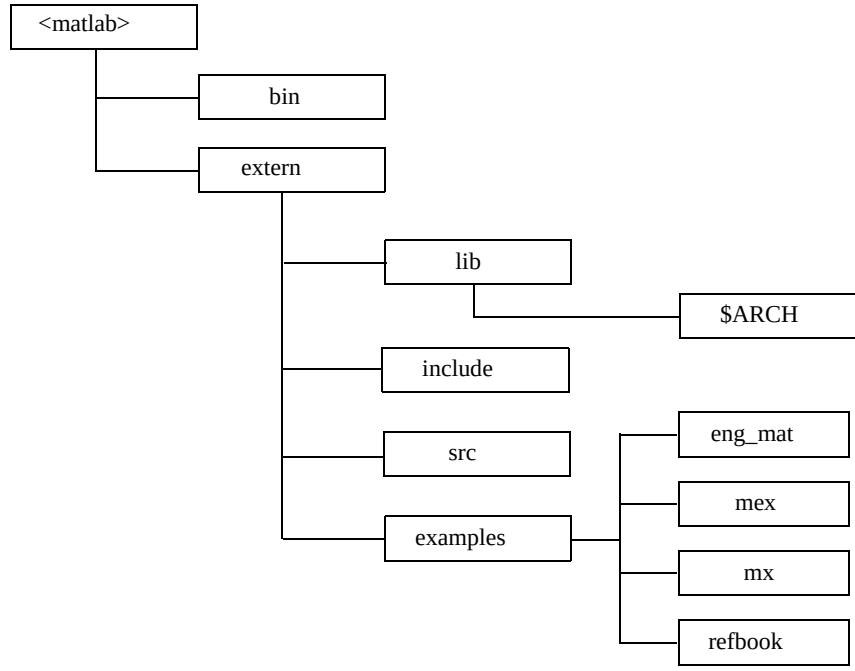
つぎの節では、ユーザアプリケーションの作成のためのその他の情報の見つけ方を説明します。

- ファイルとディレクトリ - UNIX システム
- ファイルとディレクトリ - Windows システム
- 例題
- テクニカルサポート

ファイルとディレクトリ - UNIX システム

この節では、UNIX システムにおいて、MATLAB API に関連するファイルの構成と目的を説明します。

つぎの図は、MATLAB API ファイルが存在するディレクトリを示しています。図中で、<matlab> は、システム上で MATLAB がインストールされている最上位ディレクトリを示します。



<matlab>/bin

<matlab>/bin ディレクトリは、MATLAB API に関連する 2 つのファイルを含んでいます。

mex	C または Fortran MEX- ファイルソースコードから MEX- ファイルを作成する UNIX シェルスクリプト。mex に関する詳細は、「MEX- ファイルの紹介」を参照してください。
matlab	環境を初期化し MATLAB インタプリタを呼び出す UNIX シェルスクリプト。

このディレクトリは、mex スクリプトが特定のコンパイラで用いるあらかじめ設定されたオプションファイルも含んでいます。つぎの表は、オプションファイルの一覧です。

表 1-4: あらかじめ設定されているオプションファイル

オプションファイル	説明
cxxopts.sh	mex スクリプトとシステムの C++ コンパイラを用いて、MEX- ファイルをコンパイルします。
engopts.sh	mex スクリプトとシステムの C または Fortran コンパイラを用いて、エンジンアプリケーションをコンパイルします。
gccopts.sh	mex スクリプトと GNU C (gcc) コンパイラを用いて、MEX- ファイルをコンパイルします。
matopts.sh	mex スクリプトとシステムの C または Fortran コンパイラを用いて、MAT- ファイルアプリケーションをコンパイルします。
mexopts.sh	mex スクリプトとシステムの ANSI C または Fortran コンパイラを用いて、MEX- ファイルをコンパイルします

<matlab>/extern/lib/$ARCH

<matlab>/extern/lib/*$ARCH* ディレクトリは、ライブラリを含みます。$ARCH は、特定の UNIX プラットフォームを指定します。UNIX プラットフォームの中には、このディレクトリは 2 つのバージョンのライブラリを含むものがあります。.a で終わるライブラリのファイル名はスタティックライブラリで、.so または .sl で終わるファイル名は共有ライブラリです。

<matlab>/extern/include

<matlab>/extern/include ディレクトリは、MATLAB とのインタフェースを行う C および C++ アプリケーションの開発用のヘッダファイルを含んでいます。

MATLAB API に関連するヘッダファイルは、つぎの通りです。

engine.h	MATLAB エンジンプログラム用ヘッダファイルで、エンジンルーチンに対する関数のプロトタイプを含みます。
mat.h	MAT- ファイルへのアクセスプログラム用ヘッダファイルで、mat ルーチン用の関数のプロトタイプを含みます。
matrix.h	mxArray 構造体の定義と行列アクセスルーチンの関数プロトタイプを含むヘッダファイルです。
mex.h	MEX- ファイルの作成用ヘッダファイルで、mex ルーチンの関数プロトタイプを含みます。

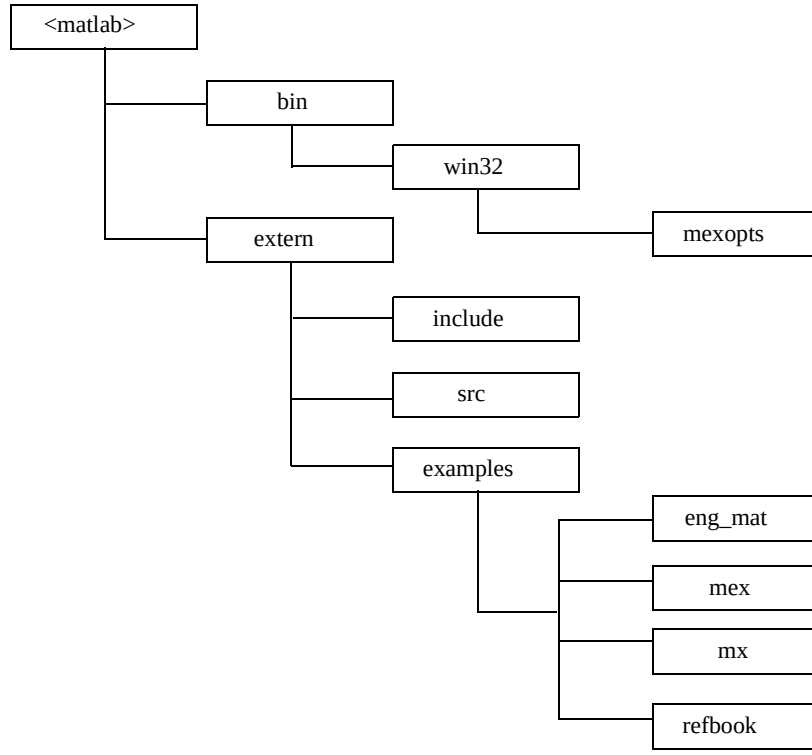
<matlab>/extern/src

<matlab>/extern/src ディレクトリは、引数チェックやバージョン設定のような MEX- ファイルのサポートに必要な C のソースファイルを含みます。

ファイルとディレクトリ - Windows システム

この節では、Microsoft Windows システムにおいて、MATLAB API に関連するファイルの構成と目的を説明します。

つぎの図は、MATLAB API ファイルが存在するディレクトリを示しています。図中で、<matlab> は、システム上で MATLAB がインストールされている最上位ディレクトリを示します。



<matlab>\bin\win32

<matlab>\bin\win32 ディレクトリは、C および Fortran ファイルから MEX- ファイルをビルドする mex.bat バッチファイルを含みます。また、このディレクトリは、mex.bat が用いる Perl スクリプト mex.pl も含みます。

<matlab>\bin\win32\mexopts

<matlab>\bin\win32\mexopts ディレクトリは、mex スクリプトが特定のコンパイラで利用するあらかじめ設定されたオプションファイルを含みます。オプションファイルの一覧は、「オプションファイル」を参照してください。

<matlab>\extern\include

<matlab>\extern\include ディレクトリは、MATLAB とのインタフェースを行う C および C++ アプリケーションの開発用のヘッダファイルを含んでいます。

MATLAB API（MEX- ファイル、エンジン、MAT- ファイル）に関連するヘッダファイルは、つぎの通りです。

engine.h	MATLAB エンジンプログラム用ヘッダファイルで、エンジンルーチンに対する関数のプロトタイプを含みます。
mat.h	MAT- ファイルへのアクセスプログラム用ヘッダファイルで、mat ルーチン用の関数のプロトタイプを含みます。
matrix.h	mxArray 構造体の定義と行列アクセスルーチンの関数プロトタイプを含むヘッダファイルです。
mex.h	MEX- ファイルの作成用ヘッダファイルで、mex ルーチンの関数プロトタイプを含みます。
_*.def	Borland compiler で用いられるファイルです。
*.def	MSVC および Microsoft Fortran コンパイラで用いられるファイルです。
mexversion.rc	MEX- ファイルにバージョン設定情報を挿入するリソースファイルです。

<matlab>\extern\src

<matlab>\extern\src ディレクトリは、MEX- ファイルのデバッグに用いるファイルを含みます。

例題

本マニュアルでは、多くの例題を使って C および Fortran MEX- ファイルの作成方法を説明しています。

テキストからの例題

extern/examples ディレクトリの refbook サブディレクトリには、本マニュアルの「*MATLAB からの C および Fortran プログラムの呼び出し*」で利用する MEX- ファイルの例題 (C および Fortran) が含まれています。

anonymous FTP サーバから、これらの例題の最新バージョンを見ることができます。

ftp://ftp.mathworks.com/pub/tech-support/docexamples/apiguide/R12/refbook

MEX リファレンスの例題

/extern/examples ディレクトリの mex サブディレクトリは、MEX- ファイルの例題を含みます。MEX インタフェース関数（接頭語 mex で始まる関数）のオンライン External Interfaces/API リファレンスページに記述されている例題を含みます。

anonymous FTP サーバから、これらの例題の最新バージョンを見ることができます。

ftp://ftp.mathworks.com/pub/tech-support/docexamples/apiguide/R12/mx

MX の例題

extern/examples の mx サブディレクトリは、配列アクセス関数の利用の例題を含んでいます。これらの関数はスタンドアロンプログラムで利用することができますが、これらのうちのほとんどは、MEX- ファイルの例です。例外は、mxSetAllocFcns.c で、この関数は、スタンドアロンプログラムでのみ利用可能です。

anonymous FTP サーバから、これらの例題の最新バージョンを見ることができます。

ftp://ftp.mathworks.com/pub/tech-support/docexamples/apiguide/R12/mex

エンジンと MAT ファイルの例題

extern/examples ディレクトリの eng_mat サブディレクトリは、MATLAB エンジンの利用のための MEX- ファイルの例題（C および Fortran）と、MATLAB データファイル（MAT- ファイル）の読み書きのための例題を含みます。これらの例題は、すべてスタンドアロンプログラムです。

anonymous FTP サーバから、これらの例題の最新バージョンを見ることができます。

ftp://ftp.mathworks.com/pub/tech-support/docexamples/apiguide/R12/eng_mat

テクニカルサポート

The MathWorks は、ウェブサイトにおいて、Technical Support を提供しています。提供されているサービスを以下に示します。

- FAQ (Frequently Asked Questions)

これは、Technical Support スタッフが頻繁に受ける質問とソリューションデータベースからの回答の一覧です。

http://www.mathworks.com/support/index.shtml

- Solution Search Engine

このウェブサイト上の knowledge base には、Technical Notes の回答とリンクが含まれ、週に数回アップデートされています。

http://www.mathworks.com/support/solutions/index.shtml

- Technical Notes

Technical notes は、一般的な質問について、Technical Support スタッフが説明しています。

http://www.mathworks.com/support/tech-notes/index.shtml

2

C 言語 MEX- ファイルの作成

本章では、C 言語で MEX- ファイルを作成する方法を説明します。MEX- ファイル本体、これらの C 言語ファイルが MATLAB とどのように作用するか、異なるデータタイプの引数をどのように渡して操作するか、MEX- ファイルプログラムのデバッグ方法、その他のアドバンスドトピックスについて説明します。

つぎの一覧は、本章の内容をまとめたものです。

- C MEX- ファイル
- C MEX- ファイルの例
- アドバンスドトピックス
- C 言語 MEX- ファイルのデバッグ

C MEX- ファイル

C MEX- ファイルは、API ルーチンの呼び出しを使って C ソースコードをコンパイルするために、mex スクリプトを使って作られます。

C MEX- ファイルの部分

MEX- ファイルのソースコードは、2 つの部分から成り立ちます。

- MEX- ファイルでインプリメントされたユーザの希望する計算を実行するコードを含む*計算ルーチン* 。計算は、データの入出力や数値計算です。
- エントリポイント mexFunction とパラメータ prhs, nrhs, plhs, nlhs により、計算ルーチンと MATLAB のインタフェースを行う*ゲートウェイルーチン*。ここで、prhs は右辺入力引数の配列、nrhs は右辺入力引数の数、plhs は左辺出力引数の配列、nlhs は左辺出力引数の数です。ゲートウェイは、計算ルーチンをサブルーチンとして呼び出します。

ゲートウェイルーチンでは、mxArray 構造体のデータにアクセスでき、このデータを C の計算ルーチン内で操作します。たとえば、mxGetPr(prhs[0]) は、prhs[0] で示された mxArray の実数データへのタイプ double * のポインタを出力します。それから、このポインタを C の他のタイプ double * のポインタのように使うことができます。ゲートウェイからユーザの C 計算ルーチンをコールした後で、出力するデータへのタイプ mxArray のポインタを設定できます。それで、MATLAB はユーザの計算ルーチンからの出力を MEX- ファイルからの出力と認識できます。

つぎの図「C MEX サイクル」は、どのように入力が MEX- ファイルに入るか、ゲートウェイ関数がどの関数を実行するか、どのように MATLAB に出力するかを示します。

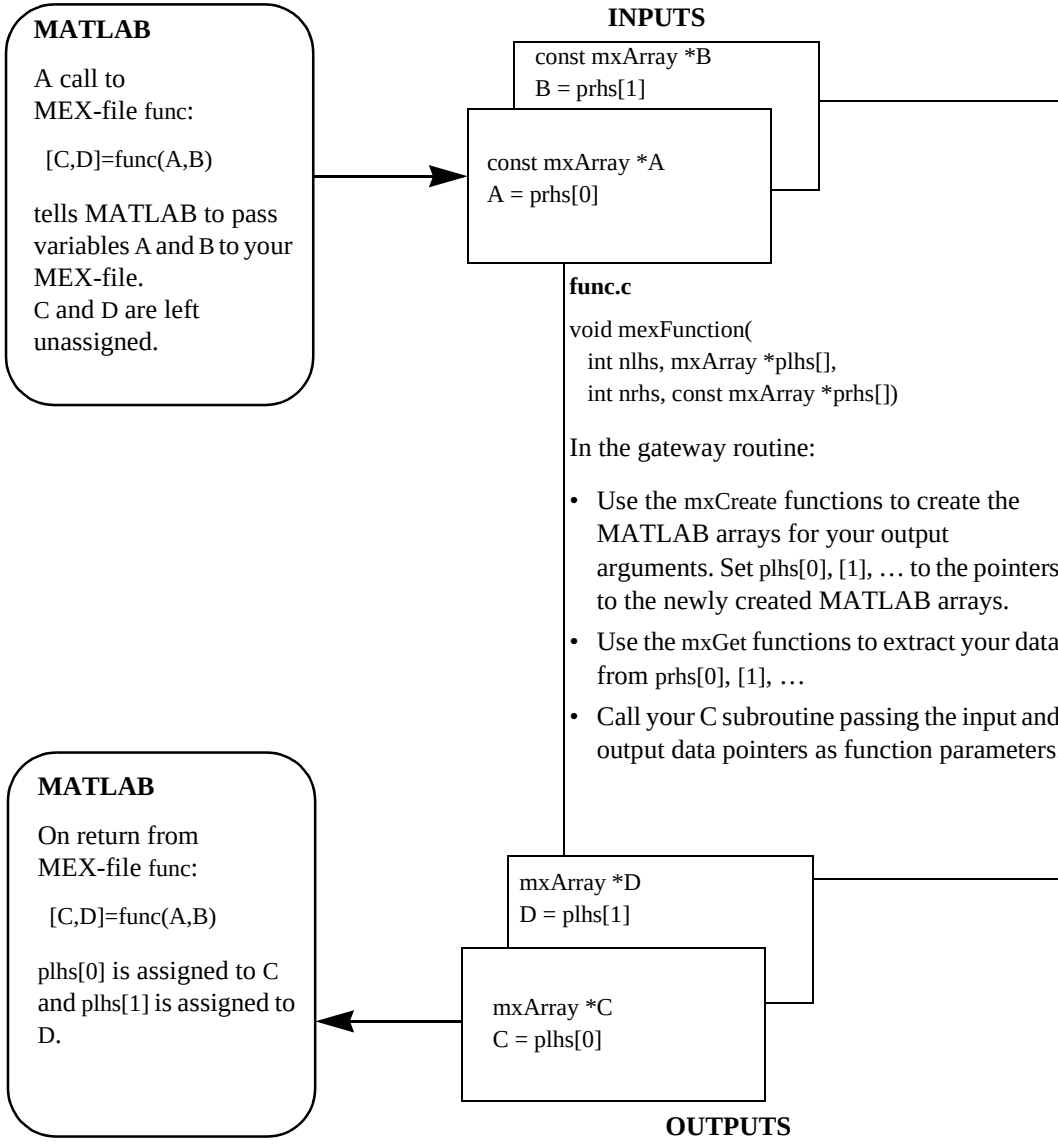


図 2-1: C MEX サイクル

MEX- ファイルに必要な引数

MEX- ファイルの 2 つの成分は分離しているか、結合しています。いずれの場合でも、エントリポイントとインタフェースルーチンが適切に宣言されるために、ファイルはヘッダ #include "mex.h" を含まなければなりません。ゲートウェイルーチン名は常に mexFunction で、つぎのパラメータを含まなければなりません。

```
void mexFunction(
   int nlhs, mxArray *plhs[],
   int nrhs, const mxArray *prhs[])
{
  /* more C code ... */
```

パラメータ nlhs と nrhs は、MEX- ファイルが呼び込まれる左辺と右辺の引数の数を含みます。MATLAB 言語のシンタックスでは、関数は一般につぎの形式です。

```
[a,b,c,…] = fun(d,e,f,…)
```

ここで、省略形 (...) は同じ形式の項目の追加を意味します。a,b,c,... は左辺引数で、d,e,f,... は右辺引数です。

パラメータ plhs と prhs は、MEX- ファイルの左辺と右辺の引数へのポインタを含むベクトルです。両者ともタイプ mxArray * と宣言され、ポイントされた変数が MATLAB配列であることを意味します。prhsはMEX-ファイルへの右辺入力へのポインタを含む長さ nrhs の配列で、plhs はユーザの関数が作る左辺出力へのポインタを含む長さ nlhs の配列です。

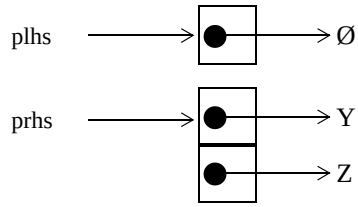
たとえば、つぎのコマンドで、MATLAB ワークスペースから MEX- ファイルを呼び込むと

```
x = fun(y,z);
```

MATLAB インタプリタはつぎの引数で mexFunction を呼び出します。

nlhs = 1

nrhs = 2



plhs は、要素が null ポインタである 1 要素の C の配列です。prhs は、1 番目の要素が Y という mxArray のポインタで、2 番目の要素が Z という mxArray のポインタである 2 要素の C の配列です。

サブルーチンが実行するまで出力 x は作成されないので、パラメータ plhs は、何もポイントしません。出力配列を作成し、plhs[0] の配列のポインタを割り当てるのはゲートウェイルーチンです。plhs[0] が割り当てられない場合は、MATLAB は出力が割り当てられていないことを示すワーニングメッセージを表示します。

注意　nlhs = 0 であっても出力値を返すことは可能です。これは、結果を ans 変数に出力することに相当します。

C MEX- ファイルの例

つぎの節は、MEX- ファイルの実行時に異なるデータタイプを渡したり、操作する方法を説明する情報や例題を含みます。トピックスは、以下の通りです。

- 第一の例 -- スカラを渡す
- 文字列を渡す
- 複数の入出力を渡す
- 構造体およびセル配列を渡す
- 複素データの操作
- 8、16、32 ビットデータの取り扱い
- 多次元数値配列の操作
- スパース配列の取り扱い
- C MEX- ファイルからの関数の呼び出し

MATLAB APIは、MATLABでサポートされている様々なデータタイプを扱うルーチンを提供します。各データタイプに対して、データ操作に用いる特定の関数があります。最初の例では、スカラ値を 2 倍にする簡単な例を示します。それから、様々なデータタイプをどのように渡して操作し、戻すか、また複数の入出力をどのように扱うかの例を述べます。最後に、MATLAB データタイプの操作を説明します。

注意　例題プログラムの最新バージョンは、anonymous FTP サーバにあります。

ftp://ftp.mathworks.com/pub/tech-support/docexamples/apiguide/R12/refbook

---

第一の例 - スカラを渡す

C コードとその等価な MEX- ファイルの簡単な例を見てみましょう。以下は、スカラを 2 倍にする C の計算関数です。

```
#include <math.h>
void timestwo(double y[], double x[])
{
  y[0] = 2.0*x[0];
  return;
```

```
}
```

以下は、MEX- ファイル形式で書かれた同じ関数です。

```
#include "mex.h"

/*
 * timestwo.c - example found in API guide
 *
 * Computational function that takes a scalar and doubles it.
 *
 * This is a MEX-file for MATLAB.
 * Copyright (c) 1984-2000 The MathWorks, Inc.
 */

/* $Revision: 1.8 $ */

void timestwo(double y[], double x[])
{
  y[0] = 2.0*x[0];
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
  double *x,*y;
  int     mrows,ncols;

  /* Check for proper number of arguments. */
  if(nrhs != 1) {
    mexErrMsgTxt("One input required.");
  } else if(nlhs > 1) {
    mexErrMsgTxt("Too many output arguments");
  }

  /* The input must be a noncomplex scalar double.*/
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
  if(!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
      !(mrows == 1 && ncols == 1) ) {
    mexErrMsgTxt("Input must be a noncomplex scalar double.");
```

```
    }

    /* Create matrix for the return argument. */
    plhs[0] = mxCreateDoubleMatrix(mrows,ncols, mxREAL);

    /* Assign pointers to each input and output. */
    x = mxGetPr(prhs[0]);
    y = mxGetPr(plhs[0]);

    /* Call the timestwo subroutine. */
    timestwo(y,x);
}
```

C では、関数引数のチェックは、コンパイル時に行われます。MATLAB では、任意のタイプおよび数の引数を M- ファンクションに渡すことが可能で、これは引数チェックが行います。MEX- ファイルに対しても同様です。ユーザプログラムは、サポートされているタイプの入力引数または出力引数の数を安全に扱う必要があります。

この例題のソースファイルをコンパイル、リンクするには、MATLAB プロンプトでつぎのようにタイプします。

```
mex timestwo.c
```

これは、実行しているプラットフォームに対応する拡張子をもつ、timestwo という MEX- ファイルを作成するのに必要なステップを行います。timestwo は M- ファンクションのように呼び出すことができます。

```
x = 2;
y = timestwo(x)
y =
    4
```

MATLABまたはオペレーティングシステムのプロンプトでMEX-ファイルを作成しコンパイルできます。MATLAB は、mex スクリプトの M- ファイル版である mex.m を使い、オペレーティングシステムは Windows では mex.bat を、UNIX では mex.sh を使います。いずれの場合でも、プロンプト

```
mex filename
```

上記の例で、スカラは 1 行 1 列の行列として表されます。

スカラ変数のコピーのポインタの代わりに、スカラ値を出力する mxGetScalar という特殊な API 関数を使うことができます。以下は、そのコードです (簡潔にするため、エラーチェックは省略されています)。

```
#include "mex.h"

/*
 * timestwoalt.c - example found in API guide
 *
 * Use mxGetScalar to return the values of scalars instead of
 * pointers to copies of scalar variables.
 *
 * This is a MEX-file for MATLAB.
 * Copyright (c) 1984-2000 The MathWorks, Inc.
 */

/* $Revision: 1.5 $ */
void timestwo_alt(double *y, double x)
{
  *y = 2.0*x;
}

void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
  double *y;
  double  x;

  /* Create a 1-by-1 matrix for the return argument. */
  plhs[0] = mxCreateDoubleMatrix(1,1,mxREAL);

  /* Get the scalar value of the input x. */
  /* Note: mxGetScalar returns a value, not a pointer. */
  x = mxGetScalar(prhs[0]);

  /* Assign a pointer to the output. */
  y = mxGetPr(plhs[0]);

  /* Call the timestwo_alt subroutine. */
  timestwo_alt(y,x);
```

```
}
```

この例は、入力スカラ x を値でサブルーチン timestwo_alt に渡しますが、出力スカラ y はポインタとして渡します。

文字列を渡す

任意の MATLAB データタイプを、MEX- ファイルで受け渡すことができます。たとえば、つぎの C コードは、文字列を受け取り、反対の順番にしてキャラクタを出力します。

```
/* $Revision: 1.10 $ */
/*=================================================================
 * revord.c
 * Example for illustrating how to copy the string data from
 * MATLAB to a C-style string and back again.
 *
 * Takes a string and returns a string in reverse order.
 *
 * This is a MEX-file for MATLAB.
 * Copyright (c) 1984-2000 The MathWorks, Inc.
 *=================================================================*/
#include "mex.h"

void revord(char *input_buf, int buflen, char *output_buf)
{
  int   i;

  /* Reverse the order of the input string. */
  for(i = 0; i < buflen-1; i++)
    *(output_buf+i) = *(input_buf+buflen-i-2);
}
```

この例題では、ダイナミックなメモリ割り当てに C の標準関数 calloc の代わりに API 関数 mxCalloc を使います。mxCalloc は、MATLAB のメモリマネージャを使ってダイナミックなメモリ割り当てを行い、ゼロに初期化します。C で calloc を使う必要がある場合には、mxCalloc を使わなければなりません。mxMalloc と mxRealloc に対しても同様です。C で malloc を使う必要がある場合は mxMalloc を使い、realloc を使う必要がある場合は mxRealloc を使ってください。

注意　　MATLAB は、MEX- ファイルを終了するときに mx 割り当てルーチン（mxCalloc, mxMalloc, mxRealloc）によって割り当てられたメモリを自動的に開放します。開放したくない場合は、API 関数 mexMakeMemoryPersistent を使ってください。

下記は、C 計算ルーチン revord を呼び出すゲートウェイ関数です。

```
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
  char *input_buf, *output_buf;
  int   buflen,status;

  /* Check for proper number of arguments. */
  if(nrhs != 1)
    mexErrMsgTxt("One input required.");
  else if(nlhs > 1)
    mexErrMsgTxt("Too many output arguments.");

  /* Input must be a string. */
  if(mxIsChar(prhs[0]) != 1)
    mexErrMsgTxt("Input must be a string.");

  /* Input must be a row vector. */
  if(mxGetM(prhs[0]) != 1)
    mexErrMsgTxt("Input must be a row vector.");

  /* Get the length of the input string. */
  buflen = (mxGetM(prhs[0]) * mxGetN(prhs[0])) + 1;

  /* Allocate memory for input and output strings. */
  input_buf = mxCalloc(buflen, sizeof(char));
  output_buf = mxCalloc(buflen, sizeof(char));

  /* Copy the string data from prhs[0] into a C string
   * input_buf. If the string array contains several rows,
   * they are copied, one column at a time, into one long
   * string array. */
```

```
  status = mxGetString(prhs[0], input_buf, buflen);
  if(status != 0)
    mexWarnMsgTxt("Not enough space. String is truncated.");

  /* Call the C subroutine. */
  revord(input_buf, buflen, output_buf);

  /* Set C-style string output_buf to MATLAB mexFunction output*/
  plhs[0] = mxCreateString(output_buf);
  return;
}
```

ゲートウェイルーチンは、入力および出力文字列に対してメモリを割り当てます。これらは C の文字列なので、MATLAB 文字列内の要素数よりも 1 大きい必要があります。つづいて MATLAB 文字列が入力文字列にコピーされます。入力および出力文字列は、出力を反対の順番にロードする計算サブルーチン (revord) に渡されます。mxCalloc はメモリを 0 に初期化するので、出力バッファは有効な null で終了する C の文字列であることに注意してください。API 関数 `mxCreateString` は、C の文字列 output_buf から MATLAB 文字列を作成します。最後に、左辺出力引数 plhs[0] は、ユーザが作成した MATLAB 配列に設定されます。

タイプ mxArray の変数を計算サブルーチンから分離することによって、オリジナルの C コードを大きく変更しなくてもすみます。

この例題では、

```
x = 'hello world';
y = revord(x)
```

とタイプすると、以下を出力します。

```
The string to convert is 'hello world'
y =
dlrow olleh
```

複数の入出力を渡す

パラメータ plhs[] と prhs[] は、各左辺 (出力) 変数と各右辺 (入力) 変数へのポインタを含むベクトルです。plhs[0] は、1 番目の左辺引数へのポインタを含み、plhs[1] は 2 番目の左辺引数へのポインタを含みます。同様に、prhs[0] は 1 番目の右辺引数へのポインタを含み、prhs[1] は 2 番目の右辺引数へのポインタを含みます。

この例題 xtimesy では、入力スカラに対して、入力スカラまたは行列を乗算し、行列を出力します。たとえば、2 つのスカラを与えて xtimesy を使います。

```
x = 7;
y = 7;
z = xtimesy(x,y)

z =
    49
```

スカラと行列を与えた xtimesy を使います。

```
x = 9;
y = ones(3);
z = xtimesy(x,y)

z =
    9    9    9
    9    9    9
    9    9    9
```

以下は、対応する MEX- ファイル C コードです。

```
#include "mex.h"

/*
 * xtimesy.c - example found in API guide
 *
 * Multiplies an input scalar times an input matrix and outputs a
 * matrix.
 *
 * This is a MEX-file for MATLAB.
 * Copyright (c) 1984-2000 The MathWorks, Inc.
 */

/* $Revision: 1.10 $ */
```

```
void xtimesy(double x, double *y, double *z, int m, int n)
{
  int i,j,count = 0;

  for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++) {
      *(z+count) = x * *(y+count);
      count++;
    }
  }
}

/* The gateway routine */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
  double *y,*z;
  double  x;
  int     status,mrows,ncols;

  /*  Check for proper number of arguments. */
  /* NOTE: You do not need an else statement when using
     mexErrMsgTxt within an if statement. It will never
     get to the else statement if mexErrMsgTxt is executed.
     (mexErrMsgTxt breaks you out of the MEX-file.)
  */
  if(nrhs != 2)
    mexErrMsgTxt("Two inputs required.");
  if(nlhs != 1)
    mexErrMsgTxt("One output required.");

  /* Check to make sure the first input argument is a scalar. */
  if(!mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
      mxGetN(prhs[0])*mxGetM(prhs[0]) != 1 ) {
    mexErrMsgTxt("Input x must be a scalar.");
  }

  /* Get the scalar input x. */
  x = mxGetScalar(prhs[0]);
```

```
    /* Create a pointer to the input matrix y. */
    y = mxGetPr(prhs[1]);

    /* Get the dimensions of the matrix input y. */
    mrows = mxGetM(prhs[1]);
    ncols = mxGetN(prhs[1]);

    /* Set the output pointer to the output matrix. */
    plhs[0] = mxCreateDoubleMatrix(mrows,ncols, mxREAL);

    /* Create a C pointer to a copy of the output matrix. */
    z = mxGetPr(plhs[0]);

    /* Call the C subroutine. */
    xtimesy(x,y,z,mrows,ncols);
}
```

この例題が示すように、複数の入出力を扱う MEX- ファイルゲートウェイの作成は簡単です。ベクトル prhs と plhs のどのインデックスが関数の入出力引数に対応するかを見るだけでいいのです。上記の例では、入力変数 x は prhs[0] に対応し、入力変数 y は prhs[1] に対応します。

mxGetScalarはxへのポインタではなくxの値を出力することに注意してください。これは、スカラ操作のその他の方法です。x を 1 行 1 列の行列として扱い、x のポインタを出力するために mxGetPr を使います。

構造体とセル配列を渡す

構造体とセル配列は、MATLAB 5 での新しいデータタイプです。構造体とセル配列の機能の説明や、それらの操作のための組み込み関数の説明は、「構造体とセル配列」を参照してください。MATLAB の他のすべてのデータタイプのように、構造体とセル配列は、C MEX- ファイルに渡されます。

構造体やセル配列を MEX- ファイルに渡すことは、データ自身がタイプ mxArray であることを除き、他のデータタイプを渡すことと似ています。実際には、これは mxGetField(構造体) と mxGetCell(セル配列) がタイプ mxArray のポインタを出力することを意味します。このポインタをタイプ mxArray の他のポインタのように扱うことができますが、mxArray に含まれるデータを C ルーチンに渡したい場合は、アクセスのために mxGetData のような API 関数を使う必要があります。

つぎの例は、m 行 n 列の構造体行列を入力として受け取り、以下のフィールドをもつ 1 行 1 列の構造体を出力します。

- 文字列入力は、m 行 n 列のセル配列を生成します。
- 数値入力 (非複素数、スカラ値) は、入力として int, double 等のような同じクラス ID をもつ m 行 n 列の数値のベクトルを生成します。

```
 * ===============================================================
 * phonebook.c
 * Example for illustrating how to manipulate structure and cell
 * array
 *
 * Takes a (MxN) structure matrix and returns a new structure
 * (1x1) containing corresponding fields:for string input, it
 * will be (MxN) cell array; and for numeric (noncomplex, scalar)
 * input, it will be (MxN) vector of numbers with the same
 * classID as input, such as int, double etc..
 *
 * This is a MEX-file for MATLAB.
 * Copyright (c) 1984-2000 The MathWorks, Inc.
*===============================================================*/
/* $Revision: 1.6 $ */

#include "mex.h"
#include "string.h"

#define MAXCHARS 80   /* max length of string contained in each
                         field */

/*  the gateway routine.  */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
  const char **fnames;       /* pointers to field names */
  const int  *dims;
  mxArray    *tmp, *fout;
  char       *pdata;
  int        ifield, jstruct, *classIDflags;
  int        NStructElems, nfields, ndim;
```

```
/* Check proper input and output */
if(nrhs != 1)
  mexErrMsgTxt("One input required.");
else if(nlhs > 1)
  mexErrMsgTxt("Too many output arguments.");
else if(!mxIsStruct(prhs[0]))
  mexErrMsgTxt("Input must be a structure.");

/* Get input arguments */
nfields = mxGetNumberOfFields(prhs[0]);
NStructElems = mxGetNumberOfElements(prhs[0]);

/* allocate memory  for storing classIDflags */
classIDflags = mxCalloc(nfields, sizeof(int));

/* Check empty field, proper data type, and data type
   consistency; get classID for each field. */
for(ifield = 0; ifield < nfields; ifield++) {
  for(jstruct = 0; jstruct < NStructElems; jstruct++) {
    tmp = mxGetFieldByNumber(prhs[0], jstruct, ifield);
    if(tmp == NULL) {
      mexPrintf("%s%d\t%s%d\n",
          "FIELD:", ifield+1, "STRUCT INDEX :", jstruct+1);
      mexErrMsgTxt("Above field is empty!");
    }
    if(jstruct == 0) {
      if((!mxIsChar(tmp) && !mxIsNumeric(tmp)) ||
          mxIsSparse(tmp)) {
        mexPrintf("%s%d\t%s%d\n",
            "FIELD:", ifield+1, "STRUCT INDEX :", jstruct+1);
        mexErrMsgTxt("Above field must have either "
            "string or numeric non-sparse data.");
      }
      classIDflags[ifield] = mxGetClassID(tmp);
    } else {
      if(mxGetClassID(tmp) != classIDflags[ifield]) {
        mexPrintf("%s%d\t%s%d\n",
            "FIELD:", ifield+1, "STRUCT INDEX :", jstruct+1);
```

```
        mexErrMsgTxt("Inconsistent data type in above field!");
      }
      else if(!mxIsChar(tmp) && ((mxIsComplex(tmp) ||
         mxGetNumberOfElements(tmp) != 1))) {
        mexPrintf("%s%d\t%s%d\n",
           "FIELD:", ifield+1, "STRUCT INDEX :", jstruct+1);
        mexErrMsgTxt("Numeric data in above field "
           "must be scalar and noncomplex!");
      }
    }
  }
}

/* Allocate memory  for storing pointers */
fnames = mxCalloc(nfields, sizeof(*fnames));

/* Get field name pointers */
for(ifield = 0; ifield < nfields; ifield++) {
  fnames[ifield] = mxGetFieldNameByNumber(prhs[0],ifield);
}

/* Create a 1x1 struct matrix for output */
plhs[0] = mxCreateStructMatrix(1, 1, nfields, fnames);
mxFree(fnames);
ndim = mxGetNumberOfDimensions(prhs[0]);
dims = mxGetDimensions(prhs[0]);
for(ifield = 0; ifield < nfields; ifield++) {
  /* Create cell/numeric array */
  if(classIDflags[ifield] == mxCHAR_CLASS) {
    fout = mxCreateCellArray(ndim, dims);
  } else {
    fout = mxCreateNumericArray(ndim, dims,
       classIDflags[ifield], mxREAL);
    pdata = mxGetData(fout);
  }

  /* Copy data from input structure array */
  for(jstruct = 0; jstruct < NStructElems; jstruct++) {
    tmp = mxGetFieldByNumber(prhs[0],jstruct,ifield);
```

```
    if(mxIsChar(tmp)) {
      mxSetCell(fout, jstruct, mxDuplicateArray(tmp));
    } else {
      size_t     sizebuf;
      sizebuf = mxGetElementSize(tmp);
      memcpy(pdata, mxGetData(tmp), sizebuf);
      pdata += sizebuf;
    }
   }

   /* Set each field in output structure */
   mxSetFieldByNumber(plhs[0], 0, ifield, fout);
  }
  mxFree(classIDflags);
  return;
}
```

このプログラムの動作方法を見るためには、以下の構造体を入力します。

```
friends(1).name = 'Jordan Robert';
friends(1).phone = 3386;
friends(2).name = 'Mary Smith';
friends(2).phone = 3912;
friends(3).name = 'Stacy Flora';
friends(3).phone = 3238;
friends(4).name = 'Harry Alpert';
friends(4).phone = 3077;
```

結果は以下のようになります。

```
phonebook(friends)

ans =
    name: {1x4 cell  }
   phone: [3386 3912 3238 3077]
```

複素データの操作

MATLAB の複素数データは、実部と虚部に分かれています。MATLAB API は、データの実部と虚部の (タイプdouble *の) ポインタを出力する2つの関数mxGetPr と mxGetPi を提供しています。

以下の例は、2 つの複素行ベクトルを受け取り、それらを畳込みます。

```
/* $Revision: 1.8 $ */
/*=========================================================
 * convec.c
 * Example for illustrating how to pass complex data
 * from MATLAB to C and back again
 *
 * Convolves two complex input vectors.
 *
 * This is a MEX-file for MATLAB.
 * Copyright (c) 1984-2000 The MathWorks, Inc.
 *=======================================================*/
#include "mex.h"

/* Computational subroutine */
void convec( double *xr, double *xi, int nx,
             double *yr, double *yi, int ny,
             double *zr, double *zi)
{
  int i,j;

  zr[0] = 0.0;
  zi[0] = 0.0;
  /* Perform the convolution of the complex vectors. */
  for(i = 0; i < nx; i++) {
    for(j = 0; j < ny; j++) {
      *(zr+i+j) = *(zr+i+j) + *(xr+i) * *(yr+j) - *(xi+i)
          * *(yi+j);
      *(zi+i+j) = *(zi+i+j) + *(xr+i) * *(yi+j) + *(xi+i)
          * *(yr+j);
    }
  }
}
```

以下は、この複素畳込みを呼び出すゲートウェイルーチンです。

```
/* The gateway routine. */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
  double  *xr, *xi, *yr, *yi, *zr, *zi;
  int     rows, cols, nx, ny;

  /* Check for the proper number of arguments. */
  if(nrhs != 2)
    mexErrMsgTxt("Two inputs required.");
  if(nlhs > 1)
    mexErrMsgTxt("Too many output arguments.");

  /* Check that both inputs are row vectors. */
  if(mxGetM(prhs[0]) != 1 || mxGetM(prhs[1]) != 1 )
    mexErrMsgTxt("Both inputs must be row vectors.");
  rows = 1;

  /* Check that both inputs are complex. */
  if(!mxIsComplex(prhs[0]) || !mxIsComplex(prhs[1]) )
    mexErrMsgTxt("Inputs must be complex.\n");

  /* Get the length of each input vector. */
  nx = mxGetN(prhs[0]);
  ny = mxGetN(prhs[1]);

  /* Get pointers to real and imaginary parts of the inputs. */
  xr = mxGetPr(prhs[0]);
  xi = mxGetPi(prhs[0]);
  yr = mxGetPr(prhs[1]);
  yi = mxGetPi(prhs[1]);

  /* Create a new array and set the output pointer to it. */
  cols = nx + ny - 1;
  plhs[0] = mxCreateDoubleMatrix(rows, cols, mxCOMPLEX);
  zr = mxGetPr(plhs[0]);
  zi = mxGetPi(plhs[0]);
```

```
    /* Call the C subroutine. */
    convec(xr, xi, nx, yr, yi, ny, zr, zi);

    return;
}
```

MATLAB プロンプトで以下の数値を入力します。

```
x = [3.000 - 1.000i, 4.000 + 2.000i, 7.000 - 3.000i];
y = [8.000 - 6.000i, 12.000 + 16.000i, 40.000 - 42.000i];
```

新しい MEX- ファイルを呼び込みます。

```
z = convec(x,y)
```

結果はつぎのようになります。

```
z =
   1.0e+02 *

 Columns 1 through 4

 0.1800 - 0.2600i 0.9600 + 0.2800i 1.3200 - 1.4400i 3.7600 - 0.1200i

 Column 5

 1.5400 - 4.1400i
```

これは、MATLAB 組込み関数 conv.m での結果と一致します。

8 ビット、16 ビット、32 ビットデータの操作

MEX- ファイル内で符号付きと符号なしの 8 ビット、16 ビット、32 ビットデータを作成し、操作できます。MATLAB API は、これらのデータタイプをサポートする関数を提供しています。API 関数 mxCreateNumericArray は、指定したデータサイズをもつ N 次の数値配列を作成します。MATLAB API がこれらのデータタイプをどのように表わすかについての説明は、オンラインリファレンスページの mxClassID を参照してください。

指定したデータタイプのMATLAB配列を作成すると、mxGetDataとmxGetImagDataを使ってデータにアクセスできます。これらの 2 つの関数は、実部と虚部のデータのポインタを出力します。MEX- ファイル内の 8 ビット、16 ビット、32 ビット精度のデータの数値演算が可能で、MATLAB に結果を出力し、MATLAB は正し

いデータのクラスを認識します。MATLAB の内部からでは 8 ビット、16 ビット、32 ビット精度の数値演算や、これらのデータ操作を行う MATLAB 関数を呼び出来ませんが、MATLAB プロンプトにデータを表示して MAT- ファイルに保存することはできます。

つぎの例は、符号なしの 16 ビット整数からなる 2 行 2 列の行列を作り、各要素を 2 倍し、両方の行列を MATLAB に出力します。

```
#include <string.h> /* Needed for memcpy() */
#include "mex.h"

/*
 * doubleelement.c - Example found in API Guide
 *
 * Constructs a 2-by-2 matrix with unsigned 16-bit integers,
 * doubles each element, and returns the matrix.
 *
 * This is a MEX-file for MATLAB.
 * Copyright (c) 1984-2000 The MathWorks, Inc.
 */

/* $Revision: 1.8 $ */

#define NDIMS 2
#define TOTAL_ELEMENTS 4

/* The computational subroutine */
void dbl_elem(unsigned short *x)
{
  unsigned short scalar = 2;
  int i,j;

  for(i = 0; i < 2;i++) {
    for(j = 0; j < 2;j++) {
      *(x+i+j) = scalar * *(x+i+j);
    }
  }
}

/* The gataway function */
void mexFunction( int nlhs, mxArray *plhs[],
```

```
                int nrhs, const mxArray *prhs[] )
{
  const int dims[] = {2,2};
  unsigned char *start_of_pr;
  unsigned short data[] = {1,2,3,4};
  int bytes_to_copy;

  /* Call the computational subroutine. */
  dbl_elem(data);

  /* Create a 2-by-2 array of unsigned 16-bit integers. */
  plhs[0] = mxCreateNumericArray(NDIMS,dims,
     mxUINT16_CLASS,mxREAL);

  /* Populate the real part of the created array. */
  start_of_pr = (unsigned char *)mxGetPr(plhs[0]);
  bytes_to_copy = TOTAL_ELEMENTS * mxGetElementSize(plhs[0]);
  memcpy(start_of_pr,data,bytes_to_copy);
}
```

MATLAB プロンプトで

```
doubleelement
```

と入力すると、以下を出力します。

```
ans =
   2   6
   8   4
```

この関数の出力は、符号なし 16 ビット整数からなる 2 行 2 列の行列です。この行列の内容を MATLAB で見ることができますが、データを操作できる方法はありません。

多次元数値配列の操作

多次元数値配列は、MATLAB 5 の新しいデータタイプです。多次元数値配列の機能と、それらを操作する MATLAB 組込み関数の説明は、MATLAB のトピックス「多次元配列」を参照してください。MATLAB の他のすべてのデータタイプのように、配列は C で書かれた MEX- ファイルに渡されます。オリジナルの多次元配列に保存されたデータの実部と虚部のポインタを出力するために、mxGetData と mxGetImagData を使って多次元数値配列を操作できます。

つぎの例は、double の多次元配列から、配列の非ゼロ要素に対するインデックスを出力します。

```
/*=================================================================
 * findnz.c
 * Example for illustrating how to handle N-dimensional arrays in
 * a MEX-file. NOTE: MATLAB uses 1-based indexing, C uses 0-based
 * indexing.
 *
 * Takes an N-dimensional array of doubles and returns the indices
 * for the non-zero elements in the array. findnz works
 * differently than the FIND command in MATLAB in that it returns
 * all the indices in one output variable, where the column
 * element contains the index for that dimension.
 *
 *
 * This is a MEX-file for MATLAB.
 * Copyright (c) 1984-2000 by The MathWorks, Inc.
 *
 *=================================================================*/

/* $Revision: 1.5 $ */
#include "mex.h"

/* If you are using a compiler that equates NaN to zero, you must
 * compile this example using the flag -DNAN_EQUALS_ZERO. For
 * example:
 *
 *     mex -DNAN_EQUALS_ZERO findnz.c
 *
 * This will correctly define the IsNonZero macro for your
   compiler. */

#if NAN_EQUALS_ZERO
#define IsNonZero(d) ((d) != 0.0 || mxIsNaN(d))
#else
#define IsNonZero(d) ((d) != 0.0)
#endif

void mexFunction(int nlhs, mxArray *plhs[],
```

```
                int nrhs, const mxArray *prhs[])
{
  /* Declare variables. */
  int elements, j, number_of_dims, cmplx;
  int nnz = 0, count = 0;
  double *pr, *pi, *pind;
  const int  *dim_array;

  /* Check for proper number of input and output arguments. */
  if(nrhs != 1) {
    mexErrMsgTxt("One input argument required.");
  }
  if(nlhs > 1) {
    mexErrMsgTxt("Too many output arguments.");
  }

  /* Check data type of input argument. */
  if(!(mxIsDouble(prhs[0]))) {
    mexErrMsgTxt("Input array must be of type double.");
  }

  /* Get the number of elements in the input argument. */
  elements = mxGetNumberOfElements(prhs[0]);

  /* Get the data. */
  pr = (double *)mxGetPr(prhs[0]);
  pi = (double *)mxGetPi(prhs[0]);
  cmplx = ((pi == NULL) ? 0 : 1);

  /* Count the number of non-zero elements to be able to allocate
     the correct size for output variable. */
  for(j = 0; j < elements; j++) {
    if(IsNonZero(pr[j]) || (cmplx && IsNonZero(pi[j]))) {
      nnz++;
    }
  }

  /* Get the number of dimensions in the input argument.
     Allocate the space for the return argument */
  number_of_dims = mxGetNumberOfDimensions(prhs[0]);
```

```
  plhs[0] = mxCreateDoubleMatrix(nnz, number_of_dims, mxREAL);
  pind = mxGetPr(plhs[0]);

  /* Get the number of dimensions in the input argument. */
  dim_array = mxGetDimensions(prhs[0]);

  /* Fill in the indices to return to MATLAB. This loops through
   * the elements and checks for non-zero values. If it finds a
   * non-zero value, it then calculates the corresponding MATLAB
    * indices and assigns them into the output array. The 1 is added
   * to the calculated index because MATLAB is 1-based and C is
   * 0-based. */
  for(j = 0; j < elements; j++) {
    if(IsNonZero(pr[j]) || (cmplx && IsNonZero(pi[j]))) {
      int temp = j;
      int k;
      for(k = 0; k < number_of_dims; k++) {
        pind[nnz*k+count] = ((temp % (dim_array[k])) + 1);
        temp /= dim_array[k];
      }
      count++;
    }
  }
}
```

MATLAB プロンプトでサンプルの行列を入力すると、以下のようになります。

```
matrix = [ 3 0 9 0; 0 8 2 4; 0 9 2 4; 3 0 9 3; 9 9 2 0]
matrix =
     3     0     9     0
     0     8     2     4
     0     9     2     4
     3     0     9     3
     9     9     2     0
```

つぎの例は、行列内のすべての非ゼロ要素の位置を決定します。この行列に対して MEX- ファイルを実行すると以下のように出力します。

```
nz = findnz(matrix)
nz =
    1    1
    4    1
    5    1
    2    2
    3    2
    5    2
    1    3
    2    3
    3    3
    4    3
    5    3
    2    4
    3    4
    4    4
```

スパース配列の操作

MATLAB API は、ユーザの MEX- ファイル内部からスパース配列を作成し操作できる関数を提供しています。これらの API ルーチンは、スパース配列に関連する 2 つのパラメータ ir と jc にアクセスしたり操作します。MATLAB のスパース配列の保存法は、「MATLAB 配列」を参照してください。

つぎの例は、スパース配列の作成方法を説明しています。

```
/*==========================================================
* fulltosparse.c
* This example demonstrates how to populate a sparse
* matrix.  For the purpose of this example, you must pass in a
* non-sparse 2-dimensional argument of type double.

* Comment: You might want to modify this MEX-file so that you can
* use it to read large sparse data sets into MATLAB.
*
* This is a MEX-file for MATLAB.
* Copyright (c) 1984-2000 The MathWorks, Inc.
*
```

```
*=================================================================*/

/* $Revision: 1.5 $ */

#include <math.h> /* Needed for the ceil() prototype. */
#include "mex.h"

/* If you are using a compiler that equates NaN to be zero, you
 * must compile this example using the flag  -DNAN_EQUALS_ZERO.
 * For example:
 *
 *     mex -DNAN_EQUALS_ZERO fulltosparse.c
 *
 * This will correctly define the IsNonZero macro for your C
 * compiler.
 */

#if defined(NAN_EQUALS_ZERO)
#define IsNonZero(d) ((d) != 0.0 || mxIsNaN(d))
#else
#define IsNonZero(d) ((d) != 0.0)
#endif

void mexFunction(
        int nlhs,       mxArray *plhs[],
        int nrhs, const mxArray *prhs[]
        )
{
  /* Declare variables. */
  int j,k,m,n,nzmax,*irs,*jcs,cmplx,isfull;
  double *pr,*pi,*si,*sr;
  double percent_sparse;

  /* Check for proper number of input and output arguments. */
  if(nrhs != 1) {
    mexErrMsgTxt("One input argument required.");
  }
  if(nlhs > 1) {
    mexErrMsgTxt("Too many output arguments.");
  }
```

```
/* Check data type of input argument. */
if(!(mxIsDouble(prhs[0]))) {
  mexErrMsgTxt("Input argument must be of type double.");
}

if(mxGetNumberOfDimensions(prhs[0]) != 2) {
  mexErrMsgTxt("Input argument must be two dimensional\n");
}

/* Get the size and pointers to input data. */
m  = mxGetM(prhs[0]);
n  = mxGetN(prhs[0]);
pr = mxGetPr(prhs[0]);
pi = mxGetPi(prhs[0]);
cmplx = (pi == NULL ? 0 : 1);

/* Allocate space for sparse matrix.
 * NOTE:  Assume at most 20% of the data is sparse.  Use ceil
 * to cause it to round up.
 */

percent_sparse = 0.2;
nzmax = (int)ceil((double)m*(double)n*percent_sparse);

plhs[0] = mxCreateSparse(m,n,nzmax,cmplx);
sr  = mxGetPr(plhs[0]);
si  = mxGetPi(plhs[0]);
irs = mxGetIr(plhs[0]);
jcs = mxGetJc(plhs[0]);

/* Copy nonzeros. */
k = 0;
isfull = 0;
for(j = 0; (j < n ); j++) {
  int i;
  jcs[j] = k;
  for(i = 0; (i < m ); i++) {
```

```
    if(IsNonZero(pr[i]) || (cmplx && IsNonZero(pi[i]))) {

      /* Check to see if non-zero element will fit in
       * allocated output array.  If not, increase
       * percent_sparse by 10%, recalculate nzmax, and augment
       * the sparse array.
       */
      if(k >= nzmax) {
        int oldnzmax = nzmax;
        percent_sparse += 0.1;
        nzmax = (int)ceil((double)m*(double)n*percent_sparse);

        /* Make sure nzmax increases atleast by 1. */
        if(oldnzmax == nzmax)
          nzmax++;

        mxSetNzmax(plhs[0], nzmax);
        mxSetPr(plhs[0], mxRealloc(sr, nzmax*sizeof(double)));
        if(si != NULL)
        mxSetPi(plhs[0], mxRealloc(si, nzmax*sizeof(double)));
        mxSetIr(plhs[0], mxRealloc(irs, nzmax*sizeof(int)));

        sr  = mxGetPr(plhs[0]);
        si  = mxGetPi(plhs[0]);
        irs = mxGetIr(plhs[0]);
      }
      sr[k] = pr[i];
      if(cmplx) {
        si[k] = pi[i];
      }
      irs[k] = i;
      k++;
    }
  }
  pr += m;
  pi += m;
 }
 jcs[n] = k;
}
```

MATLAB プロンプトで

```
full = eye(5)
full =
    1    0    0    0    0
    0    1    0    0    0
    0    0    1    0    0
    0    0    0    1    0
    0    0    0    0    1
```

と入力すると、5行5列のフル単位行列を作成します。フル行列に対して fulltosparse を使うと、対応するスパース行列を生成します。

```
spar = fulltosparse(full)
spar =
   (1,1)       1
   (2,2)       1
   (3,3)       1
   (4,4)       1
   (5,5)       1
```

C MEX- ファイルからの関数の呼び出し

API関数mexCallMATLABを使って、Cソースコード内からMATLAB関数、演算子、M- ファイル、その他の MEX- ファイルを呼び出すことができます。つぎの例は、mxArray を作成し、サブ関数に様々なポインタを渡してデータを取得し、mexCall-MATLAB を呼び出して正弦関数を計算し、結果をプロットします。

```
/* $Revision: 1.4 $ */
/*=========================================================
 * sincall.c
 *
 * Example for illustrating how to use mexCallMATLAB
 *
 * Creates an mxArray and passes its associated pointers (in
 * this demo, only pointer to its real part, pointer to number of
 * rows, pointer to number of columns) to subfunction fill() to
 * get data filled up, then calls mexCallMATLAB to calculate sin
 * function and plot the result.
 *
 * This is a MEX-file for MATLAB.
 * Copyright (c) 1984-2000 The MathWorks, Inc.
```

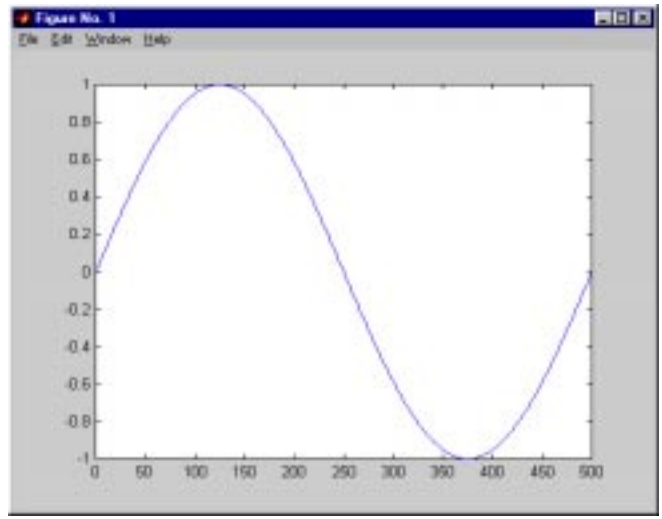
```
 *=============================================================*/
#include "mex.h"
#define MAX 1000

/* Subroutine for filling up data */
void fill( double *pr, int *pm, int *pn, int max )
{
  int i;

  /* You can fill up to max elements, so (*pr) <= max. */
  *pm = max/2;
  *pn = 1;
  for(i = 0; i < (*pm); i++)
    pr[i] = i*(4*3.14159/max);
}

/* Gateway function */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[] )
{
  int     m, n, max = MAX;
  mxArray *rhs[1], *lhs[1];

  rhs[0] = mxCreateDoubleMatrix(max, 1, mxREAL);

  /* Pass the pointers and let fill() fill up data. */
  fill(mxGetPr(rhs[0]), &m, &n, MAX);
  mxSetM(rhs[0], m);
  mxSetN(rhs[0], n);

  /* Get the sin wave and plot it. */
  mexCallMATLAB(1, lhs, 1, rhs, "sin");
  mexCallMATLAB(0, NULL, 1, lhs, "plot");

  /* Clean up allocated memory. */
  mxDestroyArray(rhs[0]);
```

```
    mxDestroyArray(lhs[0]);

    return;
}
```

例題を実行します。

```
sincall
```

結果を表示します。



注意　mexCallMATLAB を使ってタイプ mxUNKNOWN_CLASS のオブジェクトを生成できます。下記の例を参照してください。

つぎの例は、2 つの変数を出力し、それらのうちの 1 つだけに値を割り当てる M-ファイルを作成します。

```
function [a,b] = foo[c]
a = 2*c;
```

MATLAB は、以下のワーニングメッセージを表示します。

```
Warning: One or more output arguments not assigned during call to 'foo'.
```

mexCallMATLABを使ってfooを呼び出す場合、割り当てられていない出力変数は、タイプ mxUNKNOWN_CLASS です。

アドバンスドトピックス

この節では、アプリケーションが高度な MEX- ファイルを要求するときに利用できる MEX- ファイルの特徴を述べています。

ヘルプファイル

- 同じ名前の M- ファイルと MEX- ファイルが同じディレクトリにあるとき、MATLABインタプリタはMEX-ファイルを選ぶので、MEX-ファイルのふるまいを記述するために M- ファイルを使うことができます。MATLAB の help コマンドは、要求されたときに適切な M- ファイルを自動的に探して表示し、インタプリタは関数が呼び込まれたときに対応する MEX- ファイルを探して実行します。

複数ファイルのリンク

MEX- ファイルを作るときに、複数のオブジェクトファイルを結合したり、オブジェクトファイルライブラリを使うことができます。これを行うためには、ファイルを拡張子付きで、スペースで分離してリストします。たとえば、PC で、

```
mex circle.c square.obj rectangle.c shapes.lib
```

は、circle.dll という MEX- ファイルを作成するために .c, .obj, .lib ファイルに対して機能する有効なコマンドです。dll は、PC での MEX- ファイルタイプに対応する拡張子です。結果の MEX- ファイル名は、リストの 1 番目のファイルから得られます。

複数のソースファイルに関連する MEX- ファイルプロジェクトを管理するために、MAKE のようなソフトウェア開発ツールを使うと便利です。ユーザのソースファイルからオブジェクトファイルを作るルールを含む MAKEFILE を作成し、オブジェクトファイルを MEX- ファイルに統合するために mex を呼び込みます。この方法で、必要なときにのみソースファイルを確実に再コンパイルできます。

MEX- ファイル関数のワークスペース

M- ファイル関数と異なり、MEX- ファイル関数は、自身の変数ワークスペースをもちません。MEX- ファイル関数は、呼び出し側のワークスペースで機能します。

mexEvalString は、呼び出し側のワークスペースの文字列を評価します。さらに、mexGetArrayとmexPutArrayルーチンを使って呼び出し側のワークスペースから変数を取得したり設定したりできます。

メモリ管理

MEX-ファイル内でのメモリ管理は、通常のCまたはFortranアプリケーションのメモリ管理とは異なります。しかし、MEX- ファイルが MATLAB 自体のような大きいアプリケーション内に存在する必要があるため、考慮すべきことがあります。

テンポラリ配列の自動クリーンアップ

MEX- ファイルが MATLAB に戻るとき、左辺引数の形式 (plhs[] リストに含まれる mxArray) で計算の結果を MATLAB に与えます。このリストにない MEX- ファイルによって作成された mxArray は、自動的に破棄されます。さらに、MEX- ファイルの実行中に mxCalloc, mxMalloc, mxRealloc を使って割り当てられたメモリは、自動的に開放されます。

一般に、MEX- ファイルがテンポラリ配列を破棄し、ダイナミックに割り当てられたメモリを開放することを推奨します。自動メカニズムに依存するよりも、MEX- ファイルがこのクリーンアップを行うほうが効率的です。しかし、MEX- ファイルが標準のリターンステートメントに至らない状況があります。以下の場合には標準のリターンに達しません。

- mexErrMsgTxt の呼び出しの発生
- mexCallMATLAB の呼び出しが発生し、呼び出された関数がエラーを出力するとき (MEX- ファイルは、mexSetTrapFlag を使ってそのようなエラーをトラップしますが、すべての MEX- ファイルが必ずしもエラーをトラップする必要があるわけではありません)。
- ユーザが **Ctrl-C** を使って MEX- ファイルの実行に割り込むとき。
- MEX- ファイルの実行でメモリが不足するとき。このとき、MATLAB の out-of-memory ハンドラは、直ちに MEX- ファイルを停止します。

注意深い MEX- ファイルプログラマは、最初の 2 つの場合になる前に、すべてのテンポラリ配列とメモリを確実にクリーンアップするかもしれませんが、最後の 2 つの場合はしないかもしれません。後の 2 つの場合では、メモリリークを避けるために自動クリーンアップメカニズムが必要です。

固定配列

mexMakeArrayPersistent または mexMakeMemoryPersistent を呼び出すことによって、MATLAB の自動クリーンアップから配列またはメモリの一部を外すことができます。しかし、MEX- ファイルがそのような固定オブジェクトを作成する場合、固定オブジェクトが適切に破棄される前に、MEX- ファイルが消去される場合は、メモリリークが発生するという危険があります。この発生を避けるために、固定オブジェクトを作成する MEX- ファイルは、オブジェクトを破棄する関数

mexAtExit を使って関数を登録するべきです (関数 mexAtExit を使ってその他のリソースを破棄することも可能です。たとえば、mexAtExit を使ってオープンしているファイルをクローズできます)。

たとえば、以下に示すのは固定配列を作成し、適切に破棄する簡単な MEX- ファイルです。

```
#include "mex.h"

static int initialized = 0;
static mxArray *persistent_array_ptr = NULL;

void cleanup(void) {
   mexPrintf("MEX-file is terminating, destroying array\n");
   mxDestroyArray(persistent_array_ptr);
}

void mexFunction(int nlhs,
   mxArray *plhs[],
   int nrhs,
   const mxArray *prhs[])
{
 if(!initialized) {
   mexPrintf("MEX-file initializing, creating array\n");

   /* Create persistent array and register its cleanup. */
   persistent_array_ptr = mxCreateDoubleMatrix(1, 1, mxREAL);
   mexMakeArrayPersistent(persistent_array_ptr);
   mexAtExit(cleanup);
   initialized = 1;

   /* Set the data of the array to some interesting value. */
   *mxGetPr(persistent_array_ptr) = 1.0;
 } else {
   mexPrintf("MEX-file executing; value of first array
      element is %g\n",
   *mxGetPr(persistent_array_ptr));
 }
}
```

ハイブリッド配列

mxSetPr, mxSetData, mxSetCell のような関数を使って、メモリの一部を mxArray に直接配置することができます。mxDestroyArray は、配列全体に加えてこれらの部分を破棄します。このため、破棄されない配列、すなわち mxDestroyArray を確実に呼び出さない配列を作成することができます。そのような配列は、破棄可能な部分と破棄不可能な部分の両方を含むため、ハイブリッド 配列と呼ばれます。

たとえば、自動変数に対して mxFree(または ANSI 関数 free() を呼び出すことは有効ではありません。そのため、以下のコードで pArray はハイブリッド配列です。

```
mxArray *pArray = mxCreateDoubleMatrix(0, 0, mxREAL);
double data[10];

mxSetPr(pArray, data);
mxSetM(pArray, 1);
mxSetN(pArray, 10);
```

ハイブリッド配列のその他の例は、子のうちの 1 つが参照のみの配列 (MEX- ファイルの入力のような修飾 const をもつ配列) であるセル配列または構造体です。MEX- ファイルへの入力も破棄されるため、配列は破棄できません。

ハイブリッド配列は破棄できないので、「テンポラリ配列の自動クリーンアップ」で記述した自動メカニズムによってクリーンアップできません。そこで記述したように、自動クリーンアップメカニズムは、ユーザ割り込みの場合にテンポラリ配列を破棄する唯一の方法です。そのため、*テンポラリハイブリッド配列*は無効であり MEX- ファイルがクラッシュする場合があります。

固定ハイブリッド配列は実行可能であっても、可能な限り利用を避けることを推奨します。

LAPACK および BLAS 関数の使用法

LAPACK は、MATLAB が数値線形代数に対して利用する大規模のマルチオーサ Fortran サブルーチンライブラリです。BLAS は、Basic Linear Algebra Subroutines の略で、MATLAB は行列の乗算および LAPACK ルーチン自身の高速化のために利用します。LAPACK および BLAS が提供する関数は、C MEX- ファイル内部から直接呼び出すことも可能です。

本節では、LAPACK および BLAS 関数を呼び出す MEX- ファイルの作成およびビルドの方法を説明します。以下に関する情報を提供します。

- 関数名の指定

- C から Fortran へ引数を渡す
- 複素数の扱い
- MEX- ファイルの作成

LAPACK から関数を利用する対称不定分解に関する例題も提供します。

関数名の指定

LAPACK または BLAS 関数の呼び出し時に、プラットフォームの中には呼び出しステートメント内の関数名の後にアンダーバーが必要なものがあります。

PC, IBM_RS, HP プラットフォームでは、後に続くアンダーバーを利用せずに、関数名のみを利用します。たとえば、LAPACK の関数 dgemm を呼び出すには、つぎのようにします。

```
dgemm (arg1, arg2, ..., argn);
```

SGI, LINUX, Solaris, Alpha プラットフォームでは、関数名の後にアンダーバーを付け加えます。たとえば、これらのプラットフォーム上で dgemm を呼び出すには、つぎのようにします。

```
dgemm_ (arg1, arg2, ..., argn);
```

C から Fortran へ引数を渡す

LAPACK および BLAS 関数は Fortran で書かれているため、これらの関数から、またこれらの関数へ渡される引数は、参照として渡される必要があります。つぎの例では、すべての引数を参照として渡す dgemm を呼び出します。アンパーサンド (&) は引数が既に参照であっても、引数の前に置きます。

```
#include "mex.h"

void mexFunction( int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[] )
{
  double *A, *B, *C, one = 1.0, zero = 0.0;
  int m,n,p;
  char *chn = "N";

  A = mxGetPr(prhs[0]);
  B = mxGetPr(prhs[1]);
  m = mxGetM(prhs[0]);
  p = mxGetN(prhs[0]);
  n = mxGetN(prhs[1]);
```

```
  if(p != mxGetM( prhs[1])) {
    mexErrMsgTxt("Inner dimensions of matrix multiply do not
       match");
  }

  plhs[0] = mxCreateDoubleMatrix( m, n, mxREAL );
  C = mxGetPr( plhs[0] );

  /* Pass all arguments to Fortran by reference */
  dgemm (chn, chn, &m, &n, &p, &one, A, &m, B, &p, &zero, C, &m);
}
```

複素数の扱い

MATLAB は、FORTRAN と異なる方法で複素数を格納します。MATLAB は、複素数の実部と虚部を別々に同じ長さのベクトル pr と pi に格納します。FORTRAN は、実部と虚部をインタリーブして 1 つの位置に同じ数値を格納します。

結果として、LAPACK および BLAS での MATLAB 関数と FORTRAN 関数間で交換された複素変数は、互換性がありません。MATLAB は、この非互換性を解決するために複素数のストレージ形式を変更する変換ルーチンを提供します。

入力引数

FORTRAN 関数の入力引数として渡されるすべての複素変数に対して、MATLAB 変数のストレージを FORTRAN 関数と互換に変換する必要があります。そのために、関数 mat2fort を使います。以下の例題を参照してください。

出力引数

FORTRAN 関数の出力引数として渡されるすべての複素変数に対して、以下を行う必要があります。

1. 複素変数用のストレージの割り当て時に、同じサイズの MATLAB 変数用の 2 倍の容量を実数変数に割り当てます。返される変数が 2 倍の容量を必要とする FORTRAN 形式を利用するために、これを行う必要があります。以下の例題の zout の割り当てを参照してください。

2. 変数が MATLAB に返されると、MATLAB と互換であるようにストレージを変換します。これを行うために関数 fort2mat を使ってください。

例題 - 複素変数を渡す

以下の例は、複素数の prhs[0] を入力として渡し、複素数の plhs[0] を出力として受け取る LAPACK 関数を MATLAB から呼び出す方法を示します。テンポラリ変数 zin および zout は、FORTRAN 形式で prhs[0] および plhs[0] を保持するために利用されます。

```
#include "mex.h"
#include "fort.h"      /* defines mat2fort and fort2mat */

void mexFunction( int nlhs, mxArray *plhs[], int nrhs, mxArray *prhs[] )
{
  int lda, n;
  double *zin, *zout;
  lda = mxGetM( prhs[0] );
  n = mxGetN( prhs[0] );

  /* Convert input to Fortran format */
  zin = mat2fort( prhs[0], lda, n );

  /* Allocate a real, complex, lda-by-n variable to store output */
  zout = (double *)mxCalloc( 2*lda*n );

  /* Call complex LAPACK function */
  zlapack_function( zin, &lda, &n, zout );

  /* Convert output to MATLAB format */
  plhs[0] = fort2mat( zout, lda, lda, n );

  /* Free intermediate Fortran format arrays */
  mxFree( zin );
  mxFree( zout );
}
```

MEX- ファイルの作成

本節の例題では、C MEX ファイル、myCmexFile.c を MATLAB がサポートするプラットフォームでコンパイルおよびリンクする方法を説明しています。各例題で、<matlab> は MATLAB ルートディレクトリを意味します。

PC または IBM_RS プラットフォームで C MEX-ファイルをビルドする場合は、リンクするライブラリファイルを明示的に指定する必要があります。

PC では、以下のいずれかを利用します。

```
mex myCmexFile.c <matlab>/extern/lib/win32/digital/df60/libmwlapack.lib

mex myCmexFile.c <matlab>/extern/lib/win32/microsoft/msvc60/libmwlapack.lib
```

IBM_RS では、以下の用法を利用します。

```
mex myCmexFile.c -L<matlab>/bin/ibm_rs -lmwlapack
```

それ以外のすべてのプラットフォームでは、C MEX- ファイルをビルドするのと同様に MEX- ファイルをビルドすることができます。たとえば、以下のようにします。

```
mex myCmexFile.c
```

例題 - LAPACK を使った対称不定分解

ディレクトリ <matlab>/extern/examples/refbook には、2 つの LAPACK 関数を呼び出す C MEX- ファイルの例題があります。ここで、<matlab> は、MATLAB ルートディレクトリを意味します。このファイルには 2 つのバージョンがあります。

- utdu_slv.c - 関数 zhesvx および dsysvx を呼び出し、PC, HP, IBM プラットフォームで利用可能です。
- utdu_slv_.c - 関数 zhesvx_ および dsysvx_ を呼び出し、SGI, LINUX, Solaris, Alpha プラットフォームで利用可能です。

出力ファイルは、同じディレクトリにあり、通常の MEX- ファイル拡張子をもつファイル名 xc_utdu_slv です。

M- ファイルラッパー utdusolve.m は、ヘルプを提供し、以下を呼び出すことができます。

```
X = xc_utdu_slv(A,B);
```

C 言語 MEX- ファイルのデバッグ方法

ほとんどのプラットフォームで、MATLAB 内で実行中の MEX- ファイルのデバッグが可能です。ブレークポイントの設定、変数の調査、行単位のソースコードのステッピングを含む完全なソースコードのデバッグができます。

注意　「トラブルシューティング」では、MEX- ファイルの問題についての情報が提供されています。

MATLAB 内から MEX- ファイルをデバッグするためには、まず MEX- ファイルを mex の -g オプションでコンパイルしなければなりません。

```
mex -g filename.c
```

UNIX でのデバッグ

デバッガ内から MATLAB を起動する必要があります。これを行うためには、MATLAB を起動するときに使いたいデバッガ名を -D オプションで指定してください。

つぎの例は、UNIX デバッガ dbx を使って Solaris で yprime.c をデバッグする方法を示しています。

```
unix> mex -g yprime.c
unix> matlab -Ddbx
<dbx> stop dlopen <matlab>/extern/examples/mex/yprime.mexsol
```

デバッガが MATLAB をメモリにロードすると、run コマンドを実行することにより、デバッガが起動されます。

```
<dbx> run
```

通常と同じように (直接、または他の関数やスクリプトによって) デバッグしたい MEX- ファイルを実行します。MEX- ファイルを実行する前に、デバッガに戻ります。

```
>> yprime(1,1:4)
<dbx> stop in ‘yprime.mexsol‘mexFunction
```

注意　　用いられているマークは back ticks (‘) であり、シングルコート (') ではありません。

MEX- ファイルがどこにロードされたか、または MATLAB が適切なコマンドを表示する場合には MEX- ファイル名を、デバッガに通知する必要があります。このとき、デバッグを開始する準備ができます。MEX- ファイルに対するソースコードを表示し、ブレークポイントを設定できます。ゲートウェイルーチンの先頭で停止するために、mexFunction で設定すると便利です。ブレークポイントから続けるためには、デバッガに continue コマンドを実行します。

```
<dbx> cont
```

ブレークポイントの 1 つをヒットしたら、変数を調べ、メモリを表示し、レジスタの検査を行うデバッガのすべての機能を使うことができます。デバッガの使用の情報については、デバッガのドキュメントを参照してください。

注意　　他の UNIX プラットフォームでのデバッグに関する情報は、The MathWorks Technical Support webサイト http://www.mathworks.com/supportにアクセスしてください。

Windows でのデバッグ

つぎの節では、様々なコンパイラを用いた Microsoft Windows でのデバッグ方法に関して説明します。

Microsoft Compiler

Microsoft compiler を利用する場合

1. DOS プロンプトで以下を入力することにより、Microsoft Visual Studio (Version 5 または 6) を起動します。

   ```
   msdev filename.dll
   ```

2. Microsoft 環境では、**Project** メニューから **Settings** を選択します。オープンしたウィンドウで、**Debug** タブを選択します。このオプションウィンドウは、エディットボックスを含みます。**Executable for debug session** とラベルされたエ

ディットボックスで、MATLAB がインストールされている絶対パスを入力します。それ以外のすべてのエディットボックスは空です。

3 ソースファイルをオープンし、コードのライン上でマウスを右クリックして、希望するラインにブレークポイントを設定します。

4 **Build** メニューから **Debug** を選択し **Go** をクリックします。

5 この状態でMATLABからMEX-ファイルを実行しMicrosoftのデバッグ環境を使うことができます。Microsoft 環境でのデバッグ方法に関する情報は、Microsoft Development StudioまたはMicrosoft Visual Studioのドキュメントを参照してください。

Watcom Compiler

Watcom Compiler を使っている場合、

1 DOS コマンドラインでつぎのようにタイプしてデバッガを起動します。

```
WDW
```

2 Watcom Debugger が起動し、**New Program** ウィンドウがオープンします。このウィンドウで、MATLAB の絶対パスをタイプします。たとえば、

```
c:\matlab\bin\matlab.exe
```

OK をクリックします。

3 **Break** メニューから、**On Image Load** を選択し、MEX- ファイル DLL 名を大文字でタイプします。たとえば、

```
YPRIME
```

ADD を選択し **OK** をクリックするとウィンドウがクローズします。

4 **Run** メニューから **GO** を選択します。これにより MATLAB が起動されます。

5 MATLAB の起動時に、コマンドウィンドウで MEX- ファイルがある場所にディレクトリを変更し、MEX- ファイルを実行します。つぎのようなメッセージが表示された場合は、メッセージを無視して **OK** をクリックします。

```
LDR: Automatic DLL Relocation in matlab.exe
LDR: DLL filename.dll base <number> relocated due to collision with matlab.exe
```

6 デバッグしたいファイルをオープンし、ソースコードにブレークポイントを設定します。

3

Fortran MEX-ファイルの作成

本章では、Fortran 言語で MEX- ファイルを作成する方法を説明します。MEX- ファイル本体、Fortran 言語ファイルが MATLAB とどのように作用するか、異なるデータタイプの引数をどのように渡して操作するか、MEX- ファイルプログラムのデバッグ方法、その他のアドバンスドトピックスについて説明します。

つぎの一覧は、本章の内容をまとめたものです。

- Fortran MEX- ファイル
- Fortran MEX- ファイルの例
- アドバンスドトピックス
- Fortran 言語 MEX- ファイルのデバッグ

Fortran MEX- ファイル

Fortran MEX- ファイルは、APIルーチンの呼び出しを使ってFortranソースコードをコンパイルするために、mex スクリプトを使って作られます。

Fortran の MEX- ファイルは、倍精度データと文字列のみを作成することができます (MATLABがサポートする任意のデータタイプを作成できるCの場合と異なります)。Fortran MEX- ファイルはコンパイルすると M- ファンクションのように扱うことができます。

Fortran MEX- ファイルの部分

この節では、Fortran MEX- ファイルで必要な特定の要素について説明します。Fortran MEX- ファイルのソースコードは、C MEX- ファイルのように2つの部分から成り立ちます。

- MEX- ファイルでインプリメントされたユーザの希望する計算を実行するコードを含む*計算ルーチン*。計算は、データの入出力や数値計算です。
- エントリポイントmexFunctionとパラメータprhs, nrhs, plhs, nlhsにより、計算ルーチンと MATLAB のインタフェースを行う*ゲートウェイルーチン*。ここで、prhs は右辺入力引数の配列、nrhs は右辺入力引数の数、plhs は左辺出力引数の配列、nlhs は左辺出力引数の数です。ゲートウェイは、計算ルーチンをサブルーチンとして呼び出します。

計算ルーチンとゲートウェイルーチンは、分離しているか、結合しています。つぎの図 Fortran MEX サイクルは、どのように入力が API 関数に入るか、ゲートウェイ関数がどの関数を実行するか、どのように MATLAB に出力するかを示します。

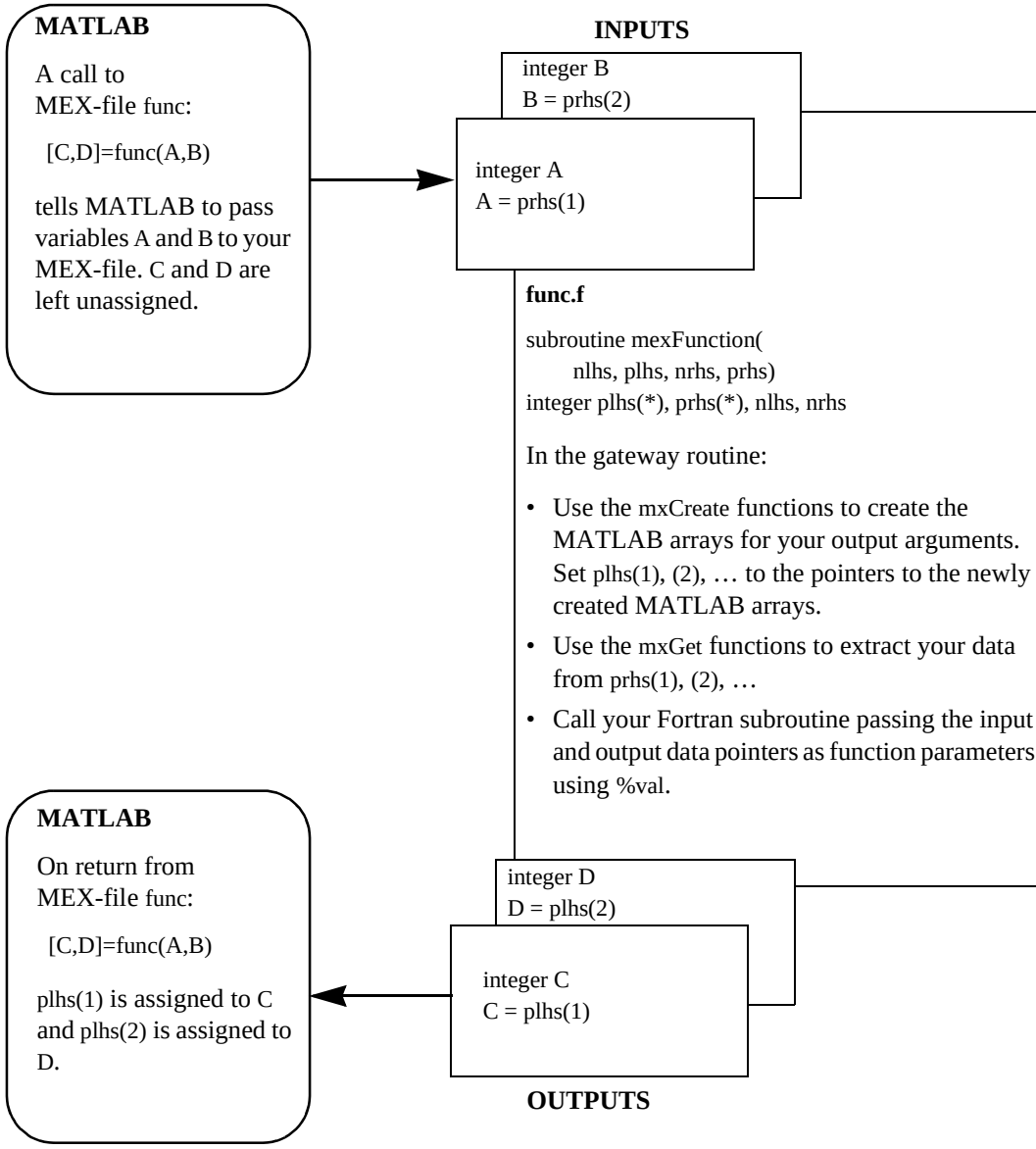


図 3-1: Fortran MEX サイクル

ポインタのコンセプト

MATLAB API は、一意的なデータタイプ mxArray をもちます。Fortran では新しいデータタイプを作成する方法がないため、MATLAB はポインタと呼ばれる特殊な識別子を Fortran プログラムに渡します。このポインタをアクセスルーチンと呼ばれる様々な API 関数に渡すことで、mxArray についての情報を得ることができます。これらのアクセスルーチンにより、mxArray のサイズ、文字列かどうか、データの内容のような、ユーザの知りたい情報を含むネイティブな Fortran のデータタイプを得ることができます。

Fortran でポインタを使うときには、つぎのような特徴があります。

- %val コンストラクト

  Fortranコンパイラが%val コンストラクトをサポートしていれば、アクセスルーチンを要求しないで使うことができるポインタ（mxGetPr または mxGetPi が出力するポインタ ）、すなわちデータのポインタがあります。このポインタの内容をサブルーチンに渡すために %val を使うことができ、これは Fortran の倍精度行列として宣言されます。

  Fortran コンパイラが %val コンストラクトをサポートしていない場合、ポインタの内容にアクセスするためには、mxCopy__ ルーチン（ 例、mxCopyPtrToReal8）を使わなければなりません。%val コンストラクトとその例に関する情報は、「%val コンストラクト」を参照してください。

- 変数の宣言

  ポインタを適切に使うために、正しいサイズになるように宣言しなければなりません。DEC Alpha マシンでは、すべてのポインタは integer*8 と宣言されます。他のすべてのプラットフォームでは、ポインタは integer*4 と宣言されます。

  Fortran コンパイラが C のプリプロセッサをサポートしていれば、プリプロセッシングステージを使ってポインタを適切な宣言へマップすることができます。UNIXでの可能な例については、examplesディレクトリの.Fで終わる例を参照してください。

注意　不正なサイズでポインタを宣言すると、プログラムのクラッシュが起こる場合があります。

ゲートウェイルーチン

ゲートウェイサブルーチンのエントリポイントは、mexFunction という名前で、つぎのパラメータを含まなければなりません。

```
subroutine mexFunction(nlhs, plhs, nrhs, prhs)
integer plhs(*), prhs(*)
integer nlhs, nrhs
```

注意　Fortran は、大文字と小文字の区別を行いません。このドキュメントは読みやすさのために、大文字と小文字の混在した関数を使います。

Fortran MEX- ファイルでは、パラメータ nlhs と nrhs は、MEX- ファイルが呼び込まれる左辺と右辺の引数の数を含みます。prhs は、MEX- ファイルの右辺入力へのポインタを含む長さ nrhs の配列で、plhs は Fortran 関数が出力する左辺出力のポインタを含む長さ nlhs の配列です。

MATLAB 言語のシンタックスでは、関数はつぎのような一般的な形式をもちます。

```
[a,b,c,…] = fun(d,e,f,…)
```

ここで、省略形 (...) は同じ形式の項を表わします。a,b,c,... は左辺引数で、d,e,f,... は右辺引数です。

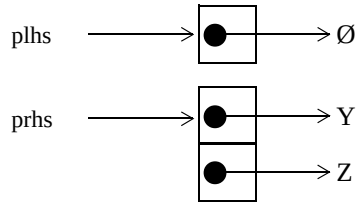
ゲートウェイルーチンの例として、MEX- ファイルをつぎのコマンドで MATLAB ワークスペースから呼び込むことを考えます。

```
x = fun(y,z);
```

MATLAB インタプリタは、つぎの引数をもつ mexFunction をコールします。

nlhs = 1

nrhs = 2



plhs は、一つの要素が null ポインタである 1 要素の C 配列です。prhs は、1 番目の要素が Y という mxArray のポインタで、2 番目の要素が Z という mxArray のポインタである 2 要素の C 配列です。

出力 x はサブルーチンが実行するまで作成されないので、パラメータ plhs は何もポイントしません。出力配列を作成し、plhs(1) の配列のポインタを設定するのはゲートウェイルーチンです。plhs(1) が割り当てられない場合は、MATLAB は出力が割り当てられていないことを示すワーニングメッセージを表示します。

注意　nlhs = 0 であっても出力値を返すことは可能です。これは、結果を ans 変数に返すことに相当します。

ゲートウェイルーチンは入力引数を確認し、誤りがあれば mexErrMsgTxt を呼び出します。これは、出力配列数のチェックと同様に、入力配列の数、タイプ、サイズのチェックを含みます。この節の後の例では、このテクニックを説明します。

mxファンクションは、MATLAB配列操作のためのアクセス法(サブルーチン)を提供します。これらの関数は、オンラインの API リファレンスページに記述されています。接頭語 mx は mxArray に対する省略表現で、関数を使って MATLAB 配列の情報の操作やアクセスができることを意味します。たとえば、mxGetPr は MATLAB配列内の実数データを取得します。MATLAB配列とFortran配列のデータの転送のためにルーチンが提供されています。

ゲートウェイルーチンは、結果を出力するために要求されるサイズの配列を作成するために、mxCreateFull, mxCreateSparse, mxCreateString を呼び出さなければなりません。これらの呼び出しの出力値は、plhs の適切な要素に割り当てられます。

ゲートウェイルーチンは、必要ならば計算ルーチン用の一時的な作業配列を割り当てるために mxCalloc を呼び出します。

ゲートウェイルーチンは、希望する計算や演算を行うために、計算ルーチンを呼び出します。MEX- ファイルが使用できる多くのルーチンがあります。これらのルーチンは、mexCallMATLAB と mexErrMsgTxt のように最初のキャラクタ mex で区別されます。

MEX- ファイルがその作業を終了するとき、MATLAB に制御が戻ります。左辺引数によって MATLAB に制御が戻らない MEX- ファイルが作成するデータ構造体は、自動的に破棄されます。

%val コンストラクト

%val コンストラクトは、ほとんどの Fortran コンパイラでサポートされています。DIGITAL Visual Fortran は、コンストラクトをサポートします。%val は、変数のアドレスではなく、変数の値をサブルーチンに渡します。%val コンストラクトをサポートしない Fortran コンパイラを使っている場合は、特殊なルーチンを使って配列の値をテンポラリな Fortran 配列にコピーしなければなりません。たとえば、計算ルーチン yprime を呼び出すゲートウェイルーチンを考えます。

```
call yprime(%val(yp), %val(t), %val(y))
```

Fortran コンパイラが %val コンストラクトをサポートしない場合、つぎのようにこの呼び出しを計算サブルーチンで置き換えます。

```
C Copy array pointers to local arrays.
      call mxCopyPtrToReal8(t, tr, 1)
      call mxCopyPtrToReal8(y, yr, 4)
C
C Call the computational subroutine.
      call yprime(ypr, tr, yr)
C
C Copy local array to output array pointer.
      call mxCopyReal8ToPtr(ypr, yp, 4)
```

ゲートウェイルーチンの最初の行につぎの宣言文を加えなければなりません。

```
real*8 ypr(4), tr, yr(4)
```

mxCopyPtrToReal8 または他の mxCopy__ ルーチンを使う場合、Fortran ゲートウェイルーチンで宣言される配列のサイズは、MATLAB からの MEX- ファイルの入力サイズ以上でなければなりません。そうでなければ、mxCopyPtrToReal8 は正常に動作しません。

Fortran MEX- ファイルの例

つぎの節は、MEX- ファイルの実行時に異なるデータタイプを渡したり、操作する方法を説明する情報や例題を含みます。トピックスは、以下の通りです。

- 第一の例 -- スカラを渡す
- 文字列を渡す
- 文字列の配列を渡す
- 行列を渡す
- 複数の入出力を渡す
- 複素データの取り扱い
- ダイナミックなメモリの割り当て
- スパース行列の取り扱い
- Fortran MEX- ファイルからの関数の呼び出し

MATLAB APIは、MATLABの倍精度データと文字列を取り扱うFortranルーチンを提供します。各データタイプに対して、データ操作に用いる特定の関数があります。

UNIX ユーザに対する注意　ディレクトリ <matlab>/extern/examples/refbook の例題の Fortran ファイルは、拡張子 .F および .f をもちます。これらの拡張子の区別として、.F ファイルは前処理を行う必要があります。

注意　例題プログラムの最新バージョンは、anonymous FTP サーバにあります。

ftp://ftp.mathworks.com/pub/tech-support/docexamples/apiguide/R12/refbook

---

第一の例 - スカラを渡す

Fortranコードとその等価なMEX- ファイルの簡単な例を見てみましょう。以下は、スカラを 2 倍する Fortran の計算ルーチンです。

```
      subroutine timestwo(y, x)
      real*8 x, y
C
      y = 2.0 * x
      return
      end
```

以下は、MEX- ファイル形式で書かれた同じ関数です。

```
C-----------------------------------------------------------------
C    timestwo.f
C
C    Multiply the input argument by 2.

C    This is a MEX-file for MATLAB.
C    Copyright (c) 1984-2000 The MathWorks, Inc.
C    $Revision: 1.11 $

      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C-----------------------------------------------------------------
C    (pointer) Replace integer by integer*8 on the DEC Alpha
C    platform.
C
      integer plhs(*), prhs(*)
      integer mxGetPr, mxCreateFull
      integer x_pr, y_pr
C-----------------------------------------------------------------
C

      integer nlhs, nrhs
      integer mxGetM, mxGetN, mxIsNumeric
      integer m, n, size
      real*8  x, y
```

```
C     Check for proper number of arguments.
      if(nrhs .ne. 1) then
        call mexErrMsgTxt('One input required.')
      elseif(nlhs .ne. 1) then
        call mexErrMsgTxt('One output required.')
      endif

C     Get the size of the input array.
      m = mxGetM(prhs(1))
      n = mxGetN(prhs(1))
      size = m*n

C     Check to ensure the input is a number.
      if(mxIsNumeric(prhs(1)) .eq. 0) then
        call mexErrMsgTxt('Input must be a number.')
      endif

C     Create matrix for the return argument.
      plhs(1) = mxCreateFull(m, n, 0)
      x_pr = mxGetPr(prhs(1))
      y_pr = mxGetPr(plhs(1))
      call mxCopyPtrToReal8(x_pr, x, size)

C     Call the computational subroutine.
      call timestwo(y, x)

C     Load the data into y_pr, which is the output to MATLAB.
      call mxCopyReal8ToPtr(y, y_pr, size)

      return
      end

      subroutine timestwo(y, x)
      real*8 x, y
C
      y = 2.0 * x
      return
      end
```

この例題のソースファイルをコンパイル、リンクするには、MATLAB プロンプトでつぎのようにタイプします。

```
mex timestwo.f
```

これは、実行しているマシンタイプに対応する拡張子をもつtimestwoというMEX-ファイルを作成するのに必要なステップを行います。timestwo を M- ファンクションのように呼び出すことができます。

```
x = 2;
y = timestwo(x)
y =
    4
```

文字列を渡す

MATLAB から Fortran MEX- ファイルに文字列を渡すことは簡単です。つぎのプログラムは、文字列を受け取りキャラクタを反対の順番にして出力します。

```
C     $Revision: 1.14 $
C=================================================================
C
C     revord.f
C     Example for illustrating how to copy string data from
C     MATLAB to a Fortran-style string and back again.
C
C     Takes a string and returns a string in reverse order.
C
C     This is a MEX-file for MATLAB.
C     Copyright (c) 1984-2000 The MathWorks, Inc.
C=================================================================

      subroutine revord(input_buf, strlen, output_buf)
      character  input_buf(*), output_buf(*)
      integer    i, strlen
      do 10 i=1,strlen
        output_buf(i) = input_buf(strlen-i+1)
 10   continue
      return
      end
```

下記は、計算ルーチンを呼び出すゲートウェイ関数です。

```
C     The gateway routine
      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
      integer nlhs, nrhs
C-----------------------------------------------------------------
C     (pointer) Replace integer by integer*8 on the DEC Alpha
C     platform.
C
      integer plhs(*), prhs(*)
      integer mxCreateString, mxGetString
C-----------------------------------------------------------------
C
      integer  mxGetM, mxGetN, mxIsString
      integer  status, strlen
      character*100 input_buf, output_buf

C     Check for proper number of arguments.
      if (nrhs .ne. 1) then
        call mexErrMsgTxt('One input required.')
      elseif (nlhs .gt. 1) then
        call mexErrMsgTxt('Too many output arguments.')
C     The input  must be a string.
      elseif(mxIsString(prhs(1)) .ne. 1) then
        call mexErrMsgTxt('Input must be a string.')
C     The input must be a row vector.
      elseif (mxGetM(prhs(1)) .ne. 1) then
        call mexErrMsgTxt('Input must be a row vector.')
      endif

C     Get the length of the input string.
      strlen = mxGetM(prhs(1))*mxGetN(prhs(1))

C     Get the string contents (dereference the input integer).
      status = mxGetString(prhs(1), input_buf, 100)

C     Check if mxGetString is successful.
      if (status .ne. 0) then
        call mexErrMsgTxt('String length must be less than 100.')
      endif
```

```
C     Initialize outbuf_buf to blanks. This is necessary on some C     compilers.

      output_buf = ' '

C     Call the computational subroutine.
      call revord(input_buf, strlen, output_buf)

C     Set output_buf to MATLAB mexFunction output.
      plhs(1) = mxCreateString(output_buf)

      return
      end
```

正しい入力数をチェックした後に、この MEX- ファイルゲートウェイルーチンは入力が行または列ベクトルの文字列であることを確認します。それから文字列のサイズを調べ、文字列を Fortran キャラクタ配列に設定します。キャラクタ文字列の場合、mxCopyPtrToCharacter を使って Fortran キャラクタ配列にデータをコピーする必要はありません。実際に、mxCopyPtrToCharacter は、MAT- ファイルにのみ動作します。MAT- ファイルについての詳細は、「データの読み込みと書き出し」を参照してください。

入力文字列

```
x = 'hello world';
```

について

```
y = revord(x)
```

とタイプすると、つぎのように出力します。

```
y =
dlrow olleh
```

文字列配列を渡す

文字列配列を渡すことは、この節の前の例「文字列を渡す」よりもわずかに複雑です。MATLAB は行の代わりに列単位として行列を保存するので、文字列配列のサイズが Fortran MEX- ファイル内で正しく定義されていることは必須です。重要な点は、MATLAB で定義される行と列のサイズが、Fortran MEX- ファイルでは反対にならなければならないことです。従って MATLAB に出力するときは、出力行列は転置されなければなりません。

つぎの例は、文字列配列やキャラクタ行列をワークスペースに直接置くのではなく、出力引数として MATLAB に置きます。MATLAB 内で、この関数を呼び出すためにつぎのようにタイプします。

```
passstr;
```

これは、5 行 15 列の行列 mystring を得ます。いくつかの操作を行う必要があります。オリジナルの文字列行列は、サイズが 5 行 15 列です。MATLAB の読み込み方法と行列内の要素の方向のために、行列のサイズは MEX ファイルで M=15 と N=5 と定義されなければなりません。行列が MATLAB 内に設定された後、行列は転置されなければなりません。

```
C    $Revision: 1.11 $
C================================================================
C
C    passstr.f
C    Example for illustrating how to pass a character matrix
C    from Fortran to MATLAB.
C
C    Passes a string array/character matrix into MATLAB as
C    output arguments rather than placing it directly into the
C    workspace.
C
C    This is a MEX-file for MATLAB.
C    Copyright (c) 1984-2000 The MathWorks, Inc.

C================================================================
     subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C-----------------------------------------------------------------
C    (pointer) Replace integer by integer*8 on the DEC Alpha
C    platform.
C
     integer plhs(*), prhs(*)
     integer p_str, mxCreateString
C-----------------------------------------------------------------
C
     integer nlhs, nrhs
     integer i
     character*75 thestring
     character*15 string(5)
```

```
C     Create the string to passed into MATLAB.
      string(1) = 'MATLAB         '
      string(2) = 'The Scientific '
      string(3) = 'Computing      '
      string(4) = 'Environment    '
      string(5) = '  by TMW, Inc.'

C     Concatenate the set of 5 strings into a long string.
      thestring = string(1)
      do 10 i = 2, 6
        thestring = thestring(:((i-1)*15)) // string(i)
 10   continue

C     Create the string matrix to be passed into MATLAB.
C     Set the matrix size to be M=15 and N=5.
      p_str = mxcreatestring(thestring)
      call mxSetM(p_str, 15)
      call mxSetN(p_str, 5)

C     Transpose the resulting matrix in MATLAB.
      call mexCallMATLAB(1, plhs, 1, p_str, 'transpose')

      return
      end
```

MATLAB プロンプトで

```
passstr
```

とタイプするとつぎの結果を生成します。

```
ans =

MATLAB
The Scientific
Computing
Environment
  by TMW, Inc.
```

行列を渡す

MATLABでは、Fortranで書かれたMEX-ファイルとの行列のやりとりができます。mxGetPrおよびmxGetPiを使ってMATLAB配列を操作し、MATLAB配列に格納されたデータの実部と虚部にポインタを割り当てます。mxCreateFullを使ってMEX-ファイル内から新規の MATLAB 配列を作成することができます。

つぎの例は、2 行 3 列の実数行列の各要素を二乗するものです。

```
C--------------------------------------------------------------
C
C    matsq.f
C
C    Squares the input matrix

C    This is a MEX-file for MATLAB.
C    Copyright (c) 1984-2000 The MathWorks, Inc.
C    $Revision: 1.12 $
C--------------------------------------------------------------

      subroutine matsq(y, x, m, n)
      real*8 x(m,n), y(m,n)
      integer m, n
C
      do 20 i=1,m
        do 10 j=1,n
          y(i,j)= x(i,j)**2
 10     continue
 20   continue
      return
      end
```

下記は、計算サブルーチンを呼び出すゲートウェイルーチンです。

```
      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C--------------------------------------------------------------
C    (pointer) Replace integer by integer*8 on the DEC Alpha
C    platform.
C
      integer plhs(*), prhs(*)
      integer mxCreateFull, mxGetPr
      integer x_pr, y_pr
```

```
C-----------------------------------------------------------------
C
      integer nlhs, nrhs
      integer mxGetM, mxGetN, mxIsNumeric
      integer m, n, size
      real*8  x(1000), y(1000)

C     Check for proper number of arguments.
      if(nrhs .ne. 1) then
        call mexErrMsgTxt('One input required.')
      elseif(nlhs .ne. 1) then
        call mexErrMsgTxt('One output required.')
      endif

C     Get the size of the input array.
      m = mxGetM(prhs(1))
      n = mxGetN(prhs(1))
      size = m*n

C     Column * row should be smaller than 1000.
      if(size.gt.1000) then
        call mexErrMsgTxt('Row * column must be <= 1000.')
      endif

C     Check to ensure the array is numeric (not strings).
      if(mxIsNumeric(prhs(1)) .eq. 0) then
        call mexErrMsgTxt('Input must be a numeric array.')
      endif

C     Create matrix for the return argument.
      plhs(1) = mxCreateFull(m, n, 0)
      x_pr = mxGetPr(prhs(1))
      y_pr = mxGetPr(plhs(1))
      call mxCopyPtrToReal8(x_pr, x, size)

C     Call the computational subroutine.
      call matsq(y, x, m, n)

C     Load the data into y_pr, which is the output to MATLAB.
      call mxCopyReal8ToPtr(y, y_pr, size)
```

```
return
end
```

正しい入出力引数がゲートウェイサブルーチンに割り当てられたことを保証し、入力が数値行列であったことを確認するエラーチェックを実行した後に、matsq.f は計算サブルーチンから出力される引数に対する行列を作成します。入力行列データは、その後 mxCopyPtrToReal8 を使って Fortran 行列にコピーされます。計算サブルーチンが呼び出され、mxCopyReal8ToPtr を使って出力引数が出力のポインタ y_pr に置かれます。

2 行 3 列の実数行列

```
x = [1 2 3; 4 5 6];
```

に対して

```
y = matsq(x)
```

はつぎの結果になります。

```
y =
     1     4     9
    16    25    36
```

複数の入出力を渡す

パラメータ plhs と prhs は、各左辺 (出力) 変数と右辺 (入力) 変数へのポインタを含むベクトルです。plhs(1) は 1 番目の左辺引数へのポインタを含み、plhs(2) は 2 番目の左辺引数へのポインタを含みます。同様に、prhs(1) は 1 番目の右辺入力引数へのポインタを含み、prhs(2) は 2 番目を意味します。

たとえば、以下のルーチンは、入力スカラまたは行列を入力スカラ回乗算します。以下は、計算サブルーチンの Fortran のコードです。

```
subroutine xtimesy(x, y, z, m, n)
real*8  x, y(3,3), z(3,3)
integer m, n
```

```
      do 20 i=1,m
        do 10 j=1,n
          z(i,j)=x*y(i,j)
10      continue
20    continue
      return
      end
```

下記は、スカラまたは行列とスカラを乗算する計算サブルーチン xtimesy を呼び出すゲートウェイルーチンです。

```
C--------------------------------------------------------------
C
C    xtimesy.f
C
C    Multiply the first input by the second input.

C    This is a MEX file for MATLAB.
C    Copyright (c) 1984-2000 The MathWorks, Inc.
C    $Revision: 1.11 $

      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C--------------------------------------------------------------
C    (pointer) Replace integer by integer*8 on the DEC Alpha
C    platform.
C
      integer plhs(*), prhs(*)
      integer mxCreateFull
      integer x_pr, y_pr, z_pr
C--------------------------------------------------------------
C
      integer nlhs, nrhs
      integer m, n, size
      integer mxGetM, mxGetN, mxIsNumeric
      real*8  x, y(3,3), z(3,3)

C     Check for proper number of arguments.
      if (nrhs .ne. 2) then
        call mexErrMsgTxt('Two inputs required.')
      elseif (nlhs .ne. 1) then
        call mexErrMsgTxt('One output required.')
```

```
      endif

C     Check to see both inputs are numeric.
      if (mxIsNumeric(prhs(1)) .ne. 1) then
        call mexErrMsgTxt('Input # 1 is not a numeric.')
      elseif (mxIsNumeric(prhs(2)) .ne. 1) then
        call mexErrMsgTxt('Input #2 is not a numeric array.')
      endif

C     Check that input #1 is a scalar.
      m = mxGetM(prhs(1))
      n = mxGetN(prhs(1))
      if(n .ne. 1 .or. m .ne. 1) then
        call mexErrMsgTxt('Input #1 is not a scalar.')
      endif

C     Get the size of the input matrix.
      m = mxGetM(prhs(2))
      n = mxGetN(prhs(2))
      size = m*n

C     Create matrix for the return argument.
      plhs(1) = mxCreateFull(m, n, 0)
      x_pr = mxGetPr(prhs(1))
      y_pr = mxGetPr(prhs(2))
      z_pr = mxGetPr(plhs(1))

C     Load the data into Fortran arrays.
      call mxCopyPtrToReal8(x_pr, x, 1)
      call mxCopyPtrToReal8(y_pr, y, size)

C     Call the computational subroutine.
      call xtimesy(x, y, z, m, n)

C     Load the output into a MATLAB array.
      call mxCopyReal8ToPtr(z, z_pr, size)

      return
      end
```

この例が示すように、複数の入出力を扱う MEX- ファイルゲートウェイの作成は簡単です。ベクトル prhs と plhs のどのインデックスが関数の入出力引数に対応するかを見るだけでいいのです。この例では、入力変数 x は prhs(1) に対応し、入力変数 y は prhs(2) に対応します。

入力スカラ x と 3 行 3 列の実数行列

```
x = 3; y = ones(3);
```

に対して

```
z = xtimesy(x, y)
```

とタイプするとつぎの結果を得ます。

```
z =
   3   3   3
   3   3   3
   3   3   3
```

複素データの操作

MATLAB は、複素数倍精度データを 2 つの数値ベクトルとして格納します。1 つは実数データを含み、もう 1 つは虚数データを含みます。API は 2 つの関数、mxCopyPtrToComplex16と mxCopyComplex16ToPtrを提供し、MATLABデータをcomplex*16 Fortran 配列にコピーします。

つぎの例は、2 つの（長さ 3 の）複素ベクトルの畳込みを行います。

```
C     $Revision: 1.14 $
C=================================================================
C
C     convec.f
C     Example for illustrating how to pass complex data from
C     MATLAB to FORTRAN (using COMPLEX data type) and back
C     again.
C
C     Convolves two complex input vectors.
C
C     This is a MEX-file for MATLAB.
C     Copyright (c) 1984-2000 The MathWorks, Inc.
C=================================================================
C
```

```
C     Computational subroutine
      subroutine convec(x, y, z, nx, ny)
      complex*16 x(*), y(*), z(*)
      integer nx, ny

C     Initialize the output array.
      do 10 i=1,nx+ny-1
        z(i) = (0.0,0.0)
 10   continue

      do 30 i=1,nx
        do 20 j=1,ny
          z(i+j-1) = z(i+j-1) + x(i) * y(j)
 20     continue
 30   continue
      return
      end

C     The gateway routine.
      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
      integer nlhs, nrhs
C-----------------------------------------------------------------
C     (pointer) Replace integer by integer*8 on the DEC Alpha
C     platform.
C
      integer plhs(*), prhs(*)
      integer mxGetPr, mxGetPi, mxCreateFull
C-----------------------------------------------------------------
C
      integer mx, nx, my, ny, nz
      integer mxGetM, mxGetN, mxIsComplex
      complex*16 x(100), y(100), z(199)

C     Check for proper number of arguments.
      if (nrhs .ne. 2) then
        call mexErrMsgTxt('Two inputs required.')
      elseif (nlhs .gt. 1) then
        call mexErrMsgTxt('Too many output arguments.')
      endif
```

```
C     Check that inputs are both row vectors.
      mx = mxGetM(prhs(1))
      nx = mxGetN(prhs(1))
      my = mxGetM(prhs(2))
      ny = mxGetN(prhs(2))
      nz = nx+ny-1

C     Only handle row vector input.
      if(mx .ne. 1 .or. my .ne. 1) then
        call mexErrMsgTxt('Both inputs must be row vector.')
C     Check sizes of the two input.
      elseif(nx .gt. 100 .or. ny .gt. 100) then
        call mexErrMsgTxt('Inputs must have less than 100
                    elements.')
C     Check to see both inputs are complex.
      elseif ((mxIsComplex(prhs(1)) .ne. 1) .or. +
            (mxIsComplex(prhs(2)) .ne. 1)) then
        call mexErrMsgTxt('Inputs must be complex.')
      endif

C     Create the output array.
      plhs(1) = mxCreateFull(1, nz, 1)

C     Load the data into Fortran arrays(native COMPLEX data).
      call mxCopyPtrToComplex16(mxGetPr(prhs(1)),
                        mxGetPi(prhs(1)), x, nx)
      call mxCopyPtrToComplex16(mxGetPr(prhs(2)),
                        mxGetPi(prhs(2)), y, ny)

C     Call the computational subroutine.
      call convec(x, y, z, nx, ny)

C     Load the output into a MATLAB array.
      call mxCopyComplex16ToPtr(z,mxGetPr(plhs(1)),
                        mxGetPi(plhs(1)), nz)

      return
      end
```

MATLAB プロンプトでつぎの数値を入力します。

```
x = [3 - 1i, 4 + 2i, 7 - 3i]

x =

   3.0000 - 1.0000i   4.0000 + 2.0000i   7.0000 - 3.0000i

y = [8 - 6i, 12 + 16i, 40 - 42i]

y =

   8.0000 - 6.0000i  12.0000 +16.0000i  40.0000 -42.0000i
```

そして新しい MEX- ファイルを呼び込みます。

```
z = convec(x, y)
```

結果はつぎのようになります。

```
z =

   1.0e+02 *

  Columns 1 through 4

   0.1800 - 0.2600i   0.9600 + 0.2800i   1.3200 - 1.4400i
   3.7600 - 0.1200i

  Column 5

   1.5400 - 4.1400i
```

これは、MATLAB 組込み関数 conv.m の結果と一致します。

メモリのダイナミックな割り当て

Fortran MEX- ファイルにメモリをダイナミックに割り当てることは可能ですが、そのためには %val を使わなければなりません。つぎの例は、実数データの入力行列の各要素を 2 倍します。

```
C     $Revision: 1.12 $
C======================================================================
C
C     dblmat.f
C     Example for illustrating how to use %val.
C     Doubles the input matrix. The demo only handles real part
C     of input.
C     NOTE: If your Fortran compiler does not support %val,
C     use mxCopy_routine.
C
C     NOTE: The subroutine compute() is in the file called
C     compute.f.
C
C     This is a MEX-file for MATLAB.
C     Copyright (c) 1984-2000 The MathWorks, Inc.
C
C======================================================================

C     Gateway subroutine

      subroutine mexfunction(nlhs, plhs, nrhs, prhs)
C-----------------------------------------------------------------
C     (pointer) Replace integer by integer*8 on the DEC Alpha
C     platform.
C
      integer plhs(*), prhs(*)
      integer pr_in, pr_out
      integer mxGetPr, mxCreateFull
C-----------------------------------------------------------------
C
      integer nlhs, nrhs, mxGetM, mxGetN
      integer m_in, n_in, size

      if(nrhs .ne. 1) then
        call mexErrMsgTxt('One input required.')
      endif
      if(nlhs .gt. 1) then
        call mexErrMsgTxt('Cannot return more than one output.')
      endif
```

```
      m_in = mxGetM(prhs(1))
      n_in = mxGetN(prhs(1))
      size = m_in * n_in
      pr_in = mxGetPr(prhs(1))
C     mxCreateFull dynamically allocates memory.
      plhs(1) = mxCreateFull(m_in, n_in, 0)
      pr_out = mxGetPr(plhs(1))

C     Call the computational routine.
      call compute(%val(pr_out), %val(pr_in), size)

      return
      end


C     $Revision: 1.3 $
C================================================================
C
C     compute.f
C
C     This subroutine doubles the input matrix. Your version of
C     compute() may do whaveter you would like it to do.
C
C     This is a MEX-file for MATLAB.
C     Copyright (c) 1984-2000 The MathWorks, Inc.
C
C================================================================

C     Computational subroutine
      subroutine compute(out_mat, in_mat, size)
      integer size, i
      real*8  out_mat(*), in_mat(*)

      do 10 i=1,size
        out_mat(i) = 2*in_mat(i)
 10   continue

      return
      end
```

2 行 3 列の入力行列

```
x = [1 2 3; 4 5 6];
```

に対して

```
y = dblmat(x)
```

とタイプするとつぎのようになります。

```
y =
   2    4    6
   8   10   12
```

注意　dblmat.f の例題は、fulltosparse.f や sincall.f と同様に、コンパイラの制限のために、ゲートウェイサブルーチンと計算サブルーチンの 2 つの部分に分かれています。

スパース行列の操作

MATLAB API は、MEX- ファイル内部からスパース行列を作成し操作できる関数を提供しています。スパース行列に関連する特殊なパラメータ ir, jc, nzmax があります。これらのパラメータの使用と一般的な MATLAB のスパース行列の格納方法に関する情報は、「MATLAB 配列」を参照してください。

注意　スパース配列のインデックス付けは、1 がベースではなく、ゼロベースです。

つぎの例は、スパース行列の作成方法を示しています。

```
C    $Revision: 1.6 $
C=================================================================
C
C    fulltosparse.f
C    Example for illustrating how to populate a sparse matrix.
C    For the purpose of this example, you must pass in a
C    non-sparse 2-dimensional argument of type real double.
```

```
C
C    NOTE: The subroutine loadsparse() is in the file called
C    loadsparse.f.
C    This is a MEX-file for MATLAB.
C    Copyright (c) 1984-2000 The MathWorks, Inc.
C
C=================================================================

C    The gateway routine.
     subroutine mexFunction(nlhs, plhs, nrhs, prhs)
     integer nlhs, nrhs
C-----------------------------------------------------------------
C    (pointer) Replace integer by integer*8 on the DEC Alpha
C    64-bit platform
C
     integer plhs(*), prhs(*)
     integer mxGetPr, mxCreateSparse, mxGetIr, mxGetJc
     integer pr, sr, irs, jcs
C-----------------------------------------------------------------
C
     integer m, n, nzmax
     integer mxGetM, mxGetN, mxIsComplex, mxIsDouble
     integer loadsparse

C    Check for proper number of arguments.
     if (nrhs .ne. 1) then
       call mexErrMsgTxt('One input argument required.')
     endif
     if (nlhs .gt. 1) then
       call mexErrMsgTxt('Too many output arguments.')
     endif

C    Check data type of input argument.
     if (mxIsDouble(prhs(1)) .eq. 0) then
       call mexErrMsgTxt('Input argument must be of type double.')
     endif
     if (mxIsComplex(prhs(1)) .eq. 1) then
       call mexErrMsgTxt('Input argument must be real only')
     endif
```

```
C     Get the size and pointers to input data.
      m = mxGetM(prhs(1))
      n = mxGetN(prhs(1))
      pr = mxGetPr(prhs(1))

C     Allocate space.
C     NOTE: Assume at most 20% of the data is sparse.
      nzmax = dble(m*n) *.20 + .5
C     NOTE: The maximum number of non-zero elements cannot be less
C     than the number of columns in the matrix.
      if (n .gt. nzmax) then
        nzmax = n
      endif
      plhs(1) = mxCreateSparse(m,n,nzmax,0)
      sr = mxGetPr(plhs(1))
      irs = mxGetIr(plhs(1))
      jcs = mxGetJc(plhs(1))

C     Load the sparse data.
      if (loadsparse(%val(pr),%val(sr),%val(irs),%val(jcs),m,n,nzmax)
     +     .eq. 1) then
        call mexPrintf('Truncating output, input is > 20%% sparse')
      endif
      return
      end
```

以下は、スパースデータをもつ mxArray の要素を満たすために fulltosparse が呼び出すサブルーチンです。

```
C     $Revision: 1.4 $
C=================================================================
C
C     loadsparse.f
C     This is the subfunction called by fulltosparse that fills the
C     mxArray with the sparse data.  Your version of
C     loadsparse can operate however you would like it to on the
C     data.
```

```
C
C    This is a MEX-file for MATLAB.
C    Copyright (c) 1984-2000 The MathWorks, Inc.
C
C=============================================================

C    Load sparse data subroutine.
     function loadsparse(a,b,ir,jc,m,n,nzmax)
     integer nzmax, m, n
     integer ir(*), jc(*)
     real*8 a(*), b(*)

     integer i, j, k

C    Copy nonzeros.
     k = 1
     do 100 j=1,n
C       NOTE: Sparse indexing is zero based.
        jc(j) = k-1
        do 200 i=1,m
           if (a((j-1)*m+i).ne. 0.0) then
              if (k .gt. nzmax) then
                 jc(n+1) = nzmax
                 loadsparse = 1
                 goto 300
              endif
              b(k) = a((j-1)*m+i)
C             NOTE: Sparse indexing is zero based.
              ir(k) = i-1
              k = k+1
           endif
 200    continue
 100  continue
C    NOTE: Sparse indexing is zero based.
     jc(n+1) = k-1
     loadsparse = 0
 300  return
     end
```

MATLAB プロンプトで

```
full = eye(5)
full =
    1    0    0    0    0
    0    1    0    0    0
    0    0    1    0    0
    0    0    0    1    0
    0    0    0    0    1
```

と入力すると、5行5列のフル単位行列を作成します。フル行列に対してfulltosparseを使うと、対応するスパース行列を生成します。

```
spar = fulltosparse(full)
spar =
  (1,1)       1
  (2,2)       1
  (3,3)       1
  (4,4)       1
  (5,5)       1
```

Fortran MEX- ファイルからの関数の呼び出し

API関数mexCallMATLABを使って、Fortranソースコード内からMATLAB関数、演算子、M- ファイル、他の MEX- ファイルを呼び出すことができます。つぎの例は、mxArray を作成し、サブ関数に様々なポインタを渡してデータを取得し、mexCallMATLAB を呼び出して正弦関数を計算し、結果をプロットします。

```
C    $Revision: 1.8 $
C ================================================================
C
C    sincall.f
C
C    Example for illustrating how to use mexCallMATLAB.
C
C    Creates an mxArray and passes its associated pointers (in
C    this demo, only pointer to its real part, pointer to number
C     of rows, pointer to number of columns) to subfunction fill()
C     to get data filled up, then calls mexCallMATLAB to calculate
C    sin function and plot the result.
C
C    NOTE: The subfunction fill() is in the file called fill.f.
```

```
C
C    This is a MEX-file for MATLAB.
C    Copyright (c) 1984-2000 The MathWorks, Inc.
C
C ============================================================

C    Gateway subroutine

     subroutine mexFunction(nlhs, plhs, nrhs, prhs)
     integer nlhs, nrhs
C------------------------------------------------------------
C    (pointer) Replace integer by integer*8 on the DEC Alpha
C    platform.
C
     integer plhs(*), prhs(*)
     integer rhs(1), lhs(1)
     integer mxGetPr, mxCreateFull
C------------------------------------------------------------
C
     integer m, n, max

C    initializition
     m=1
     n=1
     max=1000

     rhs(1) = mxCreateFull(max, 1, 0)

C    Pass the pointer and variable and let fill() fill up data.
     call fill(%val(mxGetPr(rhs(1))), m, n, max)
     call mxSetM(rhs(1), m)
     call mxSetN(rhs(1), n)

     call mexCallMATLAB(1, lhs, 1, rhs, 'sin')
     call mexCallMATLAB(0, NULL, 1, lhs, 'plot')

C    Clean up the unfreed memory after calling mexCallMATLAB.
     call mxFreeMatrix(rhs(1))
     call mxFreeMatrix(lhs(1))
```

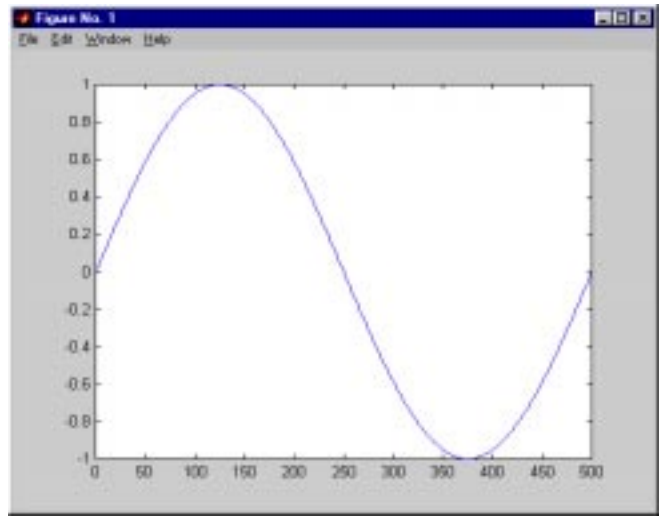
```
      return
      end


C     $Revision: 1.3 $
C ================================================================
C
C     fill.f
C     This is the subfunction called by sincall that fills the
C     mxArray with data. Your version of fill can load your data
C     however you would like.
C
C     This is a MEX-file for MATLAB.
C     Copyright (c) 1984-2000 The MathWorks, Inc.
C
C ================================================================
C     Subroutine for filling up data.

      subroutine fill(pr, m, n, max)

      real*8  pr(*)
      integer i, m, n, max

      m=max/2
      n=1
      do 10 i=1,m
 10     pr(i)=i*(4*3.1415926/max)

      return
      end
```

Fortran 計算サブルーチン内から mexCallMATLAB (または他の API ルーチン) を使うことができます。倍精度データをもつほとんどの MATLAB 関数のみ呼び出しが可能であることに注意してください。eig のような計算を実行する M- ファンクションは、倍精度でないデータに対しては正常に動作しません。

つぎの例題を実行します。

```
sincall
```

結果を表示します。



注意　mexCallMATLAB を使ってタイプ mxUNKNOWN_CLASS のオブジェクトを生成することができます。下記の例を参照してください。

つぎの例は、2 つの変数を出力し、それらのうちの 1 つだけに値を割り当てる M- ファイルを作成します。

```
function [a,b]=foo[c]
a=2*c;
```

MATLAB は、以下のワーニングメッセージを表示します。

```
Warning: One or more output arguments not assigned during call to 'foo'.
```

mexCallMATLAB を使って foo を呼び出す場合、割り当てられていない出力変数は、タイプ mxUNKNOWN_CLASS です。

アドバンスドトピックス

この節では、アプリケーションが高度な MEX- ファイルを要求するときに利用できる MEX- ファイルの特徴を述べています。

ヘルプファイル

同じ名前の M- ファイルと MEX- ファイルが同じディレクトリにあるとき、MATLAB インタプリタは MEX- ファイルを選ぶので、MEX- ファイルのふるまいを記述するために M- ファイルを使うことができます。MATLAB の help コマンドは、要求されたときに適切な M- ファイルを自動的に探して表示し、インタプリタは関数が実際に呼び込まれたときに対応する MEX- ファイルを探して実行します。

複数ファイルのリンク

MEX- ファイルを作るときに、複数のソースファイルを結合することができます。たとえば、

```
mex circle.f square.o rectangle.f shapes.o
```

は、circle.ext という MEX- ファイルを作成するために .f ファイルと .o ファイルに操作を行う有効なコマンドです。ext は、MEX- ファイルタイプに対応する拡張子です。結果の MEX- ファイル名は、リストの 1 番目のファイルから得られます。

複数のソースファイルに関連する MEX- ファイルプロジェクトを管理するために、MAKE のようなソフトウェア開発ツールを使うと便利です。ユーザのソースファイルからオブジェクトファイルを作るルールを含む MAKEFILE を作成し、それからオブジェクトファイルを MEX- ファイルに統合するために mex を呼び込みます。この方法で、必要なときにのみソースファイルを確実に再コンパイルできます。

注意　UNIX では、Fortran オブジェクトのリンクをする場合は、mex スクリプトの -fortran スイッチを使う必要があります。

MEX- ファイル関数のワークスペース

M- ファイル関数と異なり、MEX- ファイル関数は、自身の変数ワークスペースをもちません、mexEvalString は、呼び出し側のワークスペースで文字列を評価しま

す。さらに、mexGetMatrix および mexPutMatrix ルーチンを使って呼び出し側のワークスペースの変数を取得および設定することができます。

メモリ管理

Version 5.2 と同様に、MATLAB は左辺の引数リスト (plhs()) に出力されない MEX-ファイルによって作成された配列を (mxDestroyArray) を呼び出すことによって) 破棄します。その結果、MEX- ファイルの実行の終了時に残っている誤って作成された配列は、メモリエラーを起こす可能性があります。

一般に、MEX- ファイルがテンポラリ配列を破棄し、テンポラリメモリをクリーンアップすることを推奨します。メモリ管理の手法に関する情報は、「C 言語 MEX-ファイルの作成」の「メモリ管理」と「MATLAB からの C および Fortran プログラムの呼び出し」の「メモリ管理の互換性の問題」を参照してください。

Fortran 言語 MEX- ファイルのデバッグ方法

ほとんどのプラットフォームで、MATLAB 内で実行中の MEX- ファイルのデバッグが可能です。ブレークポイントの設定、変数の調査、行単位のソースコードのステッピングを含む完全なソースコードのデバッグができます。

注意　「トラブルシューティング」では、MEX- ファイルの問題についての情報が提供されています。

MATLAB 内から MEX- ファイルをデバッグするためには、まず MEX- ファイルを mex の -g オプションでコンパイルしなければなりません。

```
mex -g filename.f
```

UNIX でのデバッグ

デバッガ内から MATLAB を起動する必要があります。これを行うためには、MATLAB を起動するときに使いたいデバッガ名を -D オプションで指定してください。たとえば、UNIX デバッガ dbx を利用するには、つぎのようにタイプします。

```
matlab -Ddbx
```

デバッガが MATLAB をメモリにロードすると、run コマンドを実行することにより、デバッガが起動されます。MATLAB から MEX- ファイルのデバッグを可能にするには、

```
dbmex on
```

と MATLAB プロンプトでタイプします。それから、通常と同じように (直接、または他の関数やスクリプトによって) デバッグしたい MEX- ファイルを実行します。MEX- ファイルを実行する前に、デバッガに戻ります。

MEX- ファイルがどこにロードされたか、または MATLAB が適切なコマンドを表示する場合には MEX- ファイル名を、デバッガに通知する必要があります。このとき、デバッグを開始する準備ができます。MEX- ファイルに対するソースコードを表示し、ブレークポイントを設定できます。ゲートウェイルーチンの先頭で停止するために、mexFunction で設定すると便利です。

注意　mexFunction の名前はコンパイラによりわずかに異なる場合があります (アンダースコアが加えられている場合があります)。シンボルが MEX- ファイルでどのように現れるかを決定するためには、UNIX のコマンドを使ってください。

```
nm <MEX-file> | grep -i mexfunction
```

ブレークポイントから続けるためには、デバッガに continue コマンドを実行します。

ブレークポイントの１つをヒットしたら、変数を調べ、メモリを表示し、レジスタの検査を行うデバッガのすべての機能を使うことができます。デバッガの使用の情報については、デバッガのドキュメントを参照してください。

MATLAB プロンプトでデバッガに制御を戻したいとき、つぎのコマンドを実行します。

```
dbmex stop
```

これにより、ブレークポイントを追加したり、ソースコードを調べたりするためのデバッガへのアクセスが可能になります。実行を再開するためには、デバッガに "continue" コマンドを実行してください。

Windows でのデバッグ

DIGITAL Visual Fortran

DIGITAL Visual Fortran コンパイラを利用している場合は、プログラムのデバッグのために Microsoft debugging 環境を使います。

1. DOS プロンプトでつぎのようにタイプして Microsoft Visual Studio を起動します。

   msdev *filename*.dll

2. Microsoft 環境で、**Project** メニューから **Settings** を選択します。オープンしたウィンドウで **Debug** タブを選択します。このオプションウィンドウは、エディットボックスを含みます。**Executable for debug session** とラベルが付いたエディットボックスで、MATLAB ５がインストールされている絶対パスを入力します。その他のエディットボックスは空です。

3 ソースファイルをオープンし、コードのライン上でマウスを右クリックすることにより、希望するラインにブレークポイントを設定します。

4 **Build** メニューから **Debug** を選択し **Go** をクリックします。

5 MATLAB 内で MEX- ファイルを実行し、Microsoft debugging 環境を使うことができます。Microsoft 環境でのデバッグ方法に関する情報は、Microsoft Development Studio のドキュメントを参照してください。

4

C および Fortran プログラムからの MATLAB の呼び出し

MATLABエンジンライブラリは、ユーザプログラムからMATLABを呼び出すことができ、MATLAB を計算エンジンとして使うことができるルーチンです。MATLAB エンジンプログラムは、パイプ (UNIX)、および Windows では ActiveX を使ってMATLABプロセスと通信を行うCおよびFortranプログラムです。MATLABプロセスを起動および終了し、MATLAB とのデータのやりとりを行い、MATLABで処理されるコマンドを送信するための関数のライブラリが MATLAB で与えられています。

つぎの一覧は、本章の内容をまとめたものです。

- MATLAB エンジンの使用法
- エンジン関数の実行の例題
- エンジンプログラムのコンパイルとリンク

ユーザアプリケーションの作成における情報やサポートについては、「その他の情報」を参照してください。

MATLAB エンジンの使用法

MATLAB エンジンを使って以下のことを行うことができます。

- ユーザプログラムから配列の逆操作やFFTの計算を行う数学ルーチンを呼び出します。この用法で使われるとき、MATLAB は強力でプログラミング可能な数学サブルーチンライブラリとなります。
- たとえば、レーダーの信号解析やガスクロマトグラフィーのような、特定のタスクに対するシステムを構築します。ここで、フロントエンド (GUI) は C でプログラミングされ、バックエンド (解析) は、MATLAB でプログラミングされるので、開発時間を短縮できます。

MATLAB エンジンは、ユーザプログラムとは別のプロセスとしてバックグラウンドで実行されます。これは、つぎのような利点があります。

- UNIX では、MATLAB エンジンは、ユーザのマシン、アーキテクチャが異なるマシンを含むネットワーク上の他の UNIX マシンで実行できます。従って、ユーザのワークステーション上でユーザインタフェースをインプリメントし、ネットワーク上の他のより高速なマシンで計算を行うことができます。engOpen 関数の説明に、より詳しい情報が記述されています。
- すべてのMATLABをユーザプログラム (大量のコード) にリンクすることを要求する代わりに、少量のエンジンの通信ライブラリのみを必要とします。

エンジンライブラリ

エンジンライブラリには、MATLAB 計算エンジンを制御するための、以下のルーチンが含まれます。これらの名前は、すべて 3 文字の接頭語 eng で始まります。つぎの表は、利用可能なすべてのエンジン関数とそれらの目的の一覧です。

表 4-1: C エンジンルーチン

関数	目的
engOpen	MATLAB エンジンを起動します。
engClose	MATLAB エンジンをシャットダウンします。
engGetArray	MATLAB エンジンから MATLAB 配列を取得します。
engPutArray	MATLAB エンジンに MATLAB 配列を送信します。

表 4-1: C エンジンルーチン

関数	目的
engEvalString	MATLAB コマンドを実行します。
engOutputBuffer	MATLAB テキスト出力を格納するバッファを作成します。
engOpenSingleUse	単一の、共有しないで利用するための MATLAB セッションを起動します。

表 4-2: Fortran エンジンルーチン

関数	目的
engOpen	MATLAB エンジンを起動します。
engClose	MATLAB エンジンをシャットダウンします。
engGetMatrix	MATLAB エンジンから MATLAB 配列を取得します。
engPutMatrix	MATLAB エンジンに MATLAB 配列を送信します。
engEvalString	MATLAB コマンドを実行します。
engOutputBuffer	MATLAB テキスト出力を格納するバッファを作成します。

MATLAB エンジンは、Creating C Language MEX-Files と Creating Fortran MEX-Files で説明されている接頭語 mx が付いた API ルーチンも利用します。

MATLAB との通信

UNIX では、エンジンライブラリは、パイプと、必要ならばリモート実行のための rsh を使って、MATLAB エンジンと通信を行います。Microsoft Windows では、エンジンライブラリは、ActiveX を使って MATLAB と通信を行います。ActiveX と DDE のサポートには ActiveX の詳しい説明が含まれています。

GUI インテンシブなアプリケーション

MATLAB エンジンを通じて多くのコールバックを実行する集約的なグラフィカルユーザインタフェース (GUI) アプリケーションがある場合は、これらのコールバックをベースワークスペースのコンテキストで実行する必要があります。evalin を使って以下のようにベースワークスペースがコールバック表現の実行に利用されることを指定します。

```
engEvalString(ep, "evalin('base', expression)")
```

この方法でベースワークスペースを指定することにより、MATLAB はコールバックを正しく処理し、その呼び出しに対する結果を出力することを保証します。

これは、コールバックを実行しない計算アプリケーションには適用されません。

エンジン関数の呼び出しの例題

本節では、C および Fortran プログラムからエンジン関数を呼び出す方法を説明する例題が含まれています。例題は、以下のトピックスを説明します。

- C アプリケーションからの MATLAB の呼び出し
- Fortran アプリケーションからの MATLAB の呼び出し
- オープンされている MATLAB の利用

エンジン関数の利用時に行うべき一連の処理を理解することが重要です。たとえば、engPutArray を実行する前には、行列を作成し、名前を割り当て、要素を設定する必要があります。

これらの例題を見た後で、「エンジンプログラムのコンパイルとリンク」の指示に従って、アプリケーションを構築し、テストしてください。アプリケーションを構築し、実行することによって、システムがエンジンアプリケーションに対して適切に設定されていることが確認されます。

C アプリケーションからの MATLAB の呼び出し

このプログラム engdemo.c は、スタンドアロン C プログラムからエンジン関数を呼び出す方法を記述しています。本プログラムの Windows 版は、<matlab>\extern\examples\eng_mat ディレクトリの engwindemo.c を参照してください。MAT-ファイルの例題と同様に、エンジンの例題は eng_mat ディレクトリにあります。

```
/* $Revision: 1.6 $ */
/*
 *    engdemo.c
 *
 *    This is a simple program that illustrates how to call the
 *    MATLAB engine functions from a C program.
 *
 *     Copyright (c) 1996-2000 The MathWorks, Inc.
 *
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "engine.h"
#define  BUFSIZE 256
```

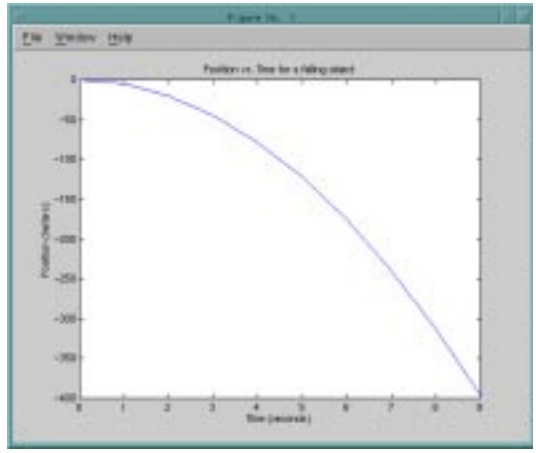
```
int main()

{
  Engine *ep;
  mxArray *T = NULL, *result = NULL;
  char buffer[BUFSIZE];
  double time[10] = { 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
     8.0, 9.0 };
  /*
   * Start the MATLAB engine locally by executing the string
   * "matlab".
   *
   * To start the session on a remote host, use the name of
   * the host as the string rather than \0.
   *
   * For more complicated cases, use any string with whitespace,
   * and that string will be executed literally to start MATLAB.
   */
  if (!(ep = engOpen("\0"))) {
    fprintf(stderr, "\nCan't start MATLAB engine\n");
    return EXIT_FAILURE;
  }

  /*
   * PART I
   *
   * For the first half of this demonstration, we will send data
   * to MATLAB, analyze the data, and plot the result.
   */

  /*
   * Create a variable for our data.
   */
  T = mxCreateDoubleMatrix(1, 10, mxREAL);
  mxSetName(T, "T");
  memcpy((void *)mxGetPr(T), (void *)time, sizeof(time));
  /*
   * Place the variable T into the MATLAB workspace.
   */
  engPutArray(ep, T);
```

```
/*
 * Evaluate a function of time, distance = (1/2)g.*t.^2
 * (g is the acceleration due to gravity).
 */
engEvalString(ep, "D = .5.*(-9.8).*T.^2;");

/*
 * Plot the result.
 */
engEvalString(ep, "plot(T,D);");
engEvalString(ep, "title('Position vs. Time for a falling
   object');");
engEvalString(ep, "xlabel('Time (seconds)');");
engEvalString(ep, "ylabel('Position (meters)');");

/*
 * Use fgetc() to make sure that we pause long enough to be
 * able to see the plot.
 */
printf("Press Return to continue\n\n");
fgetc(stdin);
/*
 * We're done for Part I! Free memory, close MATLAB engine.
 */
printf("Done for Part I.\n");
mxDestroyArray(T);
engEvalString(ep, "close;");
/*
 * PART II
 *
 * For the second half of this demonstration, we will request
 * a MATLAB string, which should define a variable X.  MATLAB
 * will evaluate the string and create the variable.  We
 * will then recover the variable, and determine its type.
 */

/*
 * Use engOutputBuffer to capture MATLAB output, so we can
 * echo it back.
```

```
 */

engOutputBuffer(ep, buffer, BUFSIZE);
while (result == NULL) {
  char str[BUFSIZE];
  /*
   * Get a string input from the user.
   */
  printf("Enter a MATLAB command to evaluate.  This
     command should\n");
  printf("create a variable X.  This program will then
     determine\n");
  printf("what kind of variable you created.\n");
  printf("For example: X = 1:5\n");
  printf(">> ");

  fgets(str, BUFSIZE-1, stdin);

  /*
   * Evaluate input with engEvalString.
   */
  engEvalString(ep, str);
  /*
   * Echo the output from the command.  First two characters
   * are always the double prompt (>>).
   */
  printf("%s", buffer+2);

  /*
   * Get result of computation.
   */
  printf("\nRetrieving X...\n");
  if ((result = engGetArray(ep,"X")) == NULL)
    printf("Oops! You didn't create a variable X.\n\n");
  else {
    printf("X is class %s\t\n", mxGetClassName(result));
  }
}
```

```
    /*
     * We're done! Free memory, close MATLAB engine and exit.
     */
    printf("Done!\n");
    mxDestroyArray(result);
    engClose(ep);

    return EXIT_SUCCESS;
}
```

このプログラムの最初の部分は、MATLAB を起動し、データを送信します。その後、MATLAB はデータを解析し、結果をプロットします。



プログラムは、つぎのように続きます。

```
Press Return to continue
```

Return を押して、プログラムを続けます。

```
Done for Part I.
Enter a MATLAB command to evaluate.  This command should
create a variable X.  This program will then determine
what kind of variable you created.
For example: X = 1:5
```

X = 17.5 を入力してプログラムの実行を続けます。

```
X = 17.5

X =

  17.5000


Retrieving X...
X is class double
Done!
```

最後に、プログラムはメモリを開放し、MATLAB エンジンをクローズして終了します。

Fortran アプリケーションからの MATLAB の呼び出し

このプログラム fngdemo.f は、スタンドアロン Fortran プログラムからエンジン関数を呼び出す方法を記述しています。

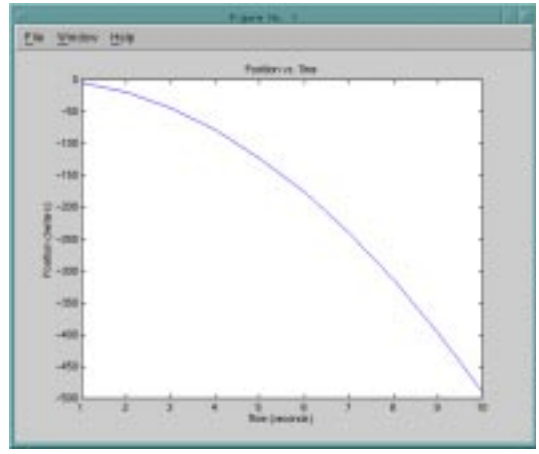
```
C
C
C     fengdemo.f
C
C     This program illustrates how to call the MATLAB
C     Engine functions from a Fortran program.
C
C Copyright (c) 1996-2000 by The MathWorks, Inc.
C
C==========================================================================
C $Revision: 1.7 $
      program main
C-----------------------------------------------------------------
C     (pointer) Replace integer by integer*8 on the DEC Alpha
C     platform.
C
      integer engOpen, engGetMatrix, mxCreateFull, mxGetPr
      integer ep, T, D
C-----------------------------------------------------------------
C
```

```
C     Other variable declarations here
      double precision time(10), dist(10)
      integer engPutMatrix, engEvalString, engClose
      integer temp, status
     data time / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0 /
C
      ep = engOpen('matlab ')
C
      if (ep .eq. 0) then
        write(6,*) 'Can't start MATLAB engine'
        stop
      endif
C
      T = mxCreateFull(1, 10, 0)
      call mxSetName(T, 'T')
      call mxCopyReal8ToPtr(time, mxGetPr(T), 10)
C
C
C     Place the variable T into the MATLAB workspace.
C
      status = engPutMatrix(ep, T)
C
      if (status .ne. 0) then
        write(6,*) 'engPutMatrix failed'
        stop
      endif
C
C
C     Evaluate a function of time, distance = (1/2)g.*t.^2
C     (g is the acceleration due to gravity)
C
      if (engEvalString(ep, 'D = .5.*(-9.8).*T.^2;') .ne. 0) then
        write(6,*) 'engEvalString failed'
        stop
      endif
C
```

```
C
C     Plot the result.
C
      if (engEvalString(ep, 'plot(T,D);') .ne. 0) then
       write(6,*) 'engEvalString failed'
         stop
      endif
     if (engEvalString(ep, 'title("Position vs. Time")') .ne. 0)      then
         write(6,*) 'engEvalString failed'
         stop
      endif
      if (engEvalString(ep, 'xlabel("Time (seconds)")') .ne. 0)
       then
         write(6,*) 'engEvalString failed'
         stop
      endif
    if (engEvalString(ep, 'ylabel("Position (meters)")') .ne. 0)     then
         write(6,*) 'engEvalString failed'
         stop
      endif
C
C
C     Read from console to make sure that we pause long enough to be
C     able to see the plot.
C
      print *, 'Type 0 <return> to Exit'
      print *, 'Type 1 <return> to continue'
      read(*,*) temp
C
      if (temp.eq.0) then
        print *, 'EXIT!'
        stop
      end if
C
      if (engEvalString(ep, 'close;') .ne. 0) then
        write(6,*) 'engEvalString failed'
        stop
      endif
C
      D = engGetMatrix(ep, 'D')
```

```
      call mxCopyPtrToReal8(mxGetPr(D), dist, 10)
      print *, 'MATLAB computed the following distances:'
      print *, '  time(s)  distance(m)'
      do 10 i=1,10
        print 20, time(i), dist(i)
 20     format(' ', G10.3, G10.3)
 10   continue
C
C
      call mxFreeMatrix(T)
      call mxFreeMatrix(D)
      status = engClose(ep)
C
      if (status .ne. 0) then
        write(6,*) 'engClose failed'
        stop
      endif
C
      stop
      end
```

プログラムを実行すると MATLAB を起動し、データを送信して結果をプロットします。



プログラムは、つぎのように続きます。

```
Type 0 <return> to Exit
Type 1 <return> to continue
```

プロンプトで 1 を入力して、プログラムの実行を続けます。

```
1
 MATLAB computed the following distances:
  time(s)  distance(m)
   1.00    -4.90
   2.00    -19.6
   3.00    -44.1
   4.00    -78.4
   5.00    -123.
   6.00    -176.
   7.00    -240.
   8.00    -314.
   9.00    -397.
   10.0    -490.
```

最後に、プログラムはメモリを開放し、MATLAB エンジンをクローズして終了します。

オープンされている MATLAB の利用

/Automation コマンドライン引数によって MATLAB を起動することにより、MATLAB エンジンプログラムは既にオープンされている MATLAB を利用できます。これにより、engOpen の呼び出しは既に起動されている MATLAB のセッションに接続されます。しかし、すべての engOpen の呼び出しが MATLAB セッションに接続されても、1 つの呼び出しに対してのみ機能します。2 つの MATLAB セッションに対して行いたい場合は、それら両方が接続されます。

/Automation 引数は、コマンドウィンドウを最小化します。マニュアルでオープンしなければなりません。

注意　/Automation コマンドライン引数と一般的な ActiveX に関する情報は、Introducing MATLAB ActiveX Integration を参照してください。

たとえば、

1 MATLAB をシャットダウンします。

2 ウィンドウのメニューバーの**スタート (Start)** ボタンからファイル名を指定して、**実行 (Run)** をクリックします。

3 **名前 (Open)** フィールドでつぎのようにタイプします。

```
d:\matlab\bin\win32\matlab.exe /Automation
```

ここで、d:\matlab\bin\win32 は、MATLAB の実行ファイルのパスを表わします。

4 **OK** をクリックすると MATLAB が起動されます。

5 MATLABで、ディレクトリを$MATLAB/extern/examples/eng_matに変更します。$MATLAB は、MATLAB のルートディレクトリです。

6 例題 engwindemo.c をコンパイルします。

7 MATLAB プロンプトでつぎのようにタイプして、engwindemo プログラムを実行します。

```
!engwindemo
```

これは、他の MATLAB プロセスを起動せずに、既にオープンされている MATLAB プロセスを使います。

注意 UNIX プラットフォームでは、MATLAB エンジンプログラムは既にオープンされている MATLAB を利用することはできません。

エンジンプログラムのコンパイルとリンク

エンジンプログラムの実行可能バージョンを生成するには、コンパイルして適切なライブラリとリンクしなければなりません。この節では、UNIX および Windows システムでエンジンプログラムをコンパイル、リンクするために必要な処理について記述します。浮動小数点の例外をマスクしないコンパイラに対して考慮すべき事柄から始めます。以下のトピックスを説明します。

- 浮動小数点の例外のマスク
- UNIX でのコンパイルとリンク
- Windows でのコンパイルとリンク

浮動小数点の例外のマスク

数学演算の中には、結果が不定値になるものがあります。たとえば、ゼロ割は IEEE の不定値 inf になります。浮動小数点の例外は、そのような演算が実行されるときに発生します。MATLAB は、inf や NaN のような不定値をサポートする IEEE モデルを使うので、MATLAB は浮動小数点の例外を利用不可能にしたり、マスクすることができます。

コンパイラの中には、デフォルトで浮動小数点の例外をマスクしないものがあります。これにより、そのようなコンパイラを使って構築されたエンジンプログラムは、浮動小数点の例外が発生したときに停止します。その結果、エンジンアプリケーションが適切に実行されるためには、これらのコンパイラを使って浮動小数点の例外をマスクする際には予防策をとることが必要です。

注意　MATLAB ベースのアプリケーションは、浮動小数点の例外が生じません。浮動小数点の例外が発生した場合は、リンクしているサードパーティのライブラリが浮動小数点の例外ハンドリングが不可能でないかを確認してください。

浮動小数点の例外をマスクするプラットフォームおよびコンパイラは、以下の通りです。

Borland C++ Compiler on Windows

Windows プラットフォームで Borland C++ compiler を利用時に浮動小数点の例外をマスクするには、ユーザプログラムにコードを追加する必要があります。MATLAB API 関数の呼び出し前に、main() または WinMain() 関数の先頭につぎのコードを追加してください。

```
#include <float.h>
      .
      .
      .
_control87(MCW_EM,MCW_EM);
      .
      .
      .
```

UNIX でのコンパイルとリンク

ランタイムの UNIX では、API 共有ライブラリの場所をシステムに通知する必要があります。この節では、ユーザのシェルおよびシステムアーキテクチャに応じて、必要な UNIX コマンドについて説明します。

ランタイムライブラリパスの設定

C シェルでは、ライブラリパスを設定するコマンドは、

```
setenv LD_LIBRARY_PATH <matlab>/extern/lib/$Arch:$LD_LIBRARY_PATH
```

Bourne シェルでは、ライブラリパスを設定するコマンドは、

```
LD_LIBRARY_PATH=<matlab>/extern/lib/$Arch:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

<matlab> は、MATLAB ルートディレクトリで、$Arch はシステムアーキテクチャ (alpha, glnx86, sgi, sol2, hp700, ibm_rs) です。

環境変数 (この例題では LD_LIBRARY_PATH) は、プラットフォームにより異なります。つぎの表は、システムで利用する環境変数名の一覧です。

表 4-3: 環境変数名

アーキテクチャ	環境変数
HP700	SHLIB_PATH
IBM RS/6000	LIBPATH

C シェルに対しては ~/.cshrc、Bourne シェルに対しては ~/.profile のように、スタートアップスクリプトにコマンドを設定すると便利です。

コンパイルとリンク

MATLAB は、エンジンアプリケーションを簡単にコンパイルおよびリンクするために、mex スクリプトを使うためのオプションファイル engopts.sh を提供しています。たとえば、例題の engdemo.c をコンパイル、リンクするには、つぎを使います。

```
mex -f <matlab>/bin/engopts.sh <pathname>/engdemo.c
```

ここで、<pathname> は指定したファイルの完全なパスを指定します。

ユーザ固有のコンパイラまたはプラットフォームに対してオプションファイルを修正する必要がある場合は、-v スイッチを使ってカレントのコンパイラとリンカの設定を表示し、その後 engopts.sh ファイルのローカルコピーに適切な変更を行ってください。

Windows でのコンパイルとリンク

エンジンプログラムをコンパイルおよびリンクするには、エンジンオプションファイルと mex スクリプトを使用します。「MEX- ファイルの作成」の表「オプションファイル」は、本リリースに含まれるエンジンオプションファイルを示しています。これらのオプションファイルは、すべて <matlab>\bin\win32\mexopts にあります。

例題として、MSVC (Version 5.0) を使って Windows でスタンドアロンエンジンアプリケーション engdemo.c をコンパイル、リンクするには、つぎのようにします。

```
mex -f <matlab>\bin\win32\mexopts\msvc50engmatopts.bat ... <pathname>\engwindemo.c
```

ここで、<pathname> は指定するファイルの完全なパスを指定します。ユーザ固有のコンパイラに対してオプションファイルを修正する必要がある場合は、-v スイッチを使ってカレントのコンパイラとリンカの設定を表示し、それからオプションファイルで適切な変更を行ってください。

5

MATLAB からの Java の呼び出し

本章では、MATLAB と Java クラスおよびオブジェクトとのインタフェースの利用法について説明します。この MATLAB の機能を利用して、Java クラスを MATLAB 環境から実行、これらのクラスからオブジェクトを作成、Java オブジェクトに対してメソッドを呼び出し、後でリロードするために Java オブジェクトを保存などが可能です。これらのすべては MATLAB 関数とコマンドによって実現されます。

つぎの一覧は、本章の内容をまとめたものです。

- MATLAB からの Java の利用： 概要
- Java クラスを MATLAB に導入する
- Java オブジェクトの作成と利用
- Java オブジェクトの実行方法
- Java 配列の利用
- Java にデータを渡す
- Java で出力されたデータの取り扱い
- プログラミングの例題

MATLAB からの Java の利用： 概要

Java インタフェースは MATLAB に不可欠です

インストールされた MATLAB には、Java Virtual Machine (JVM) が含まれているので、MATLAB コマンドを通した Java インタプリタの利用や、Java オブジェクトの作成、アクセスするプログラムを作成したり実行したりできます。MATLAB のインストールに関する情報は、プラットフォームに対応した MATLAB インストレーションドキュメントを参照してください。

MATLAB Java インタフェースの利点

MATLAB Java インタフェースを使って、以下のことを行えます。

- I/O やネットワークのような必須のアクティビティをサポートする Java API (application programming interface) クラスパッケージにアクセスします。たとえば、URL クラスは、インターネット上のリソースへアクセスするための便利な方法を提供します。
- サードパーティの Java クラスにアクセスします。
- MATLAB で Java オブジェクトを容易に作成できます。
- Java または、MATLAB シンタックスを使って、Java オブジェクトメソッドをコールします。
- MATLAB 変数と Java オブジェクト間でシームレスにデータを渡します。

MATLAB Java インタフェースを利用すべき対象

MATLAB Java インタフェースは、Java プログラミング言語の特殊機能を利用したいすべての MATLAB ユーザを対象としています。たとえば、

- MATLAB から Java クラスの機能にアクセスする必要がある。
- Java または C++ のような他の言語でのオブジェクト指向プログラミングに精通している。
- MATLAB のオブジェクト指向クラス、または、MATLAB MEX- ファイルに精通している。

Java プログラミングについてさらに学習するには

Java 言語の詳しい説明およびオブジェクト指向ソフトウェア設計とプログラミングのガイダンスについては、リソース以外を調べる必要があります。たとえば、以下の書籍が役立ちます。

- *Java in a Nutshell* (Second Edition), by David Flanagan
- *Teach Yourself Java in 21 Days*, by Lemay and Perkins

JavaSoft Web サイトから情報を入手することもできます。

```
http://www.javasoft.com
```

その他のオブジェクト指向プログラミングのリソースについては、以下を参照してください。

- *Object Oriented Software Construction*, by Bertrand Meyer
- *Object Oriented Analysis and Design with Applications*, by Grady Booch

Java Virtual Machine をサポートするプラットフォーム

プラットフォーム上で、MATLAB が使用している Java Virtual Machine (JVM) のバージョンを知るためには、MATLAB プロンプトで以下を入力してください。

```
version -java
```

Java クラスの MATLAB への導入

既存の Java クラスを導入したり、ユーザ独自のクラス定義を作成して、MATLAB を共に利用することができます。本節では、必要なクラス定義を見つける方法や、ユーザ独自の設計のクラスの作成法を説明します。個々の .class ファイルあるいは Java Archive ファイルで定義された必要なクラスを入手すると、それらを MATLAB 環境内で利用可能にする方法がわかります。

本節で注目するトピックスは、以下の通りです。

- Java クラスのソース
- 新規の Java クラスの定義
- MATLAB で利用可能な Java クラスの作成
- Java クラス定義の MATLAB へのロード
- MATLAB での Java クラス名の簡略化

Java クラスのソース

MATLAB で利用可能な Java クラスの主なソースは、3 つあります。

- **Java の組み込みクラス**

  Java 言語は、java.awt のような一般的な目的のクラスパッケージを含みます。これらのパッケージに関する説明は、Java 言語ドキュメントを参照してください。

- **サードパーティのクラス**

  ユーザが利用可能な特殊な目的のための Java クラスのパッケージが多く存在します。

- **ユーザ定義クラス**

  新規の Java クラスまたは既存のクラスのサブクラスを定義することが可能です。つぎの節で説明する Java 開発環境を使う必要があります。

新規の Java クラスの定義

新規の Java クラスや既存のクラスのサブクラスを定義するには、Java version 1.1 をサポートする、MATLAB 外部の Java 開発環境を利用する必要があります。Java Virtual Machine version 1.1.8 をサポートする開発キットは、Sun Microsystems のウェブサイト (http://www.java.sun.com/jdk/) からダウンロード可能です。また、Sun のサイトは、開発用に必要な Java 言語やクラスに関するドキュメントも提供しています。

.Java ファイルでクラス定義を作成した後で、Java コンパイラを使って .class ファイルを作成します。つぎのステップは、MATLAB で利用するために .class ファイルのクラス定義の方法を説明します。

MATLAB で利用可能な Java クラスの作成

サードパーティの Java クラスやユーザ定義の Java クラスを MATLAB で利用可能にするには、MATLAB パスにそれらを置き、ファイル classpath にそれらの位置を追加します。classpath.txt に関する情報は、「classpath.txt の検索と編集」を参照してください。

個々の（パッケージ化されていない）クラスを利用可能にする

個々のクラス（パッケージの部分でないクラス）を MATLAB で利用可能にするには、利用したい .class ファイルのディレクトリのフルパスを classpath.txt に追加します。

たとえば、ファイル d:\work\javaclasses\test.class をコンパイルした Java クラスを利用可能にするためには、つぎのエントリを classpath.txt ファイルに追加します。

```
d:\work\javaclasses
```

パッケージ全体を利用可能にする

パッケージに属する 1 つまたは複数のクラスにアクセスするには、パッケージ全体を MATLAB で利用可能にする必要があります。これは、パッケージのパスの最上位ディレクトリの親ディレクトリのフルパスをclasspath.txtに追加することにより行います。このディレクトリは、パッケージ名の最初の要素です。

たとえば、Java クラスパッケージ com.mw.tbx.ini が、ディレクトリ d:\work\com\mw\tbx\ini にクラスをもつ場合は、つぎのエントリを classpath.txt に追加します。

```
d:\work
```

JAR ファイルのクラスを利用可能にする

jarツールを使って、ZIPフォーマットで圧縮された複数のJavaクラスやパッケージを含む JAR ファイルを作成することができます。jar および JAR ファイルに関する情報は、Java 開発のドキュメントあるいは JavaSoft ウェブサイトを参照してください。また「Java プログラミングについて学習するには」も参照してください。

JAR ファイルの内容を MATLAB で利用可能にするためには、*完全なファイル名を含む* JAR ファイルのフルパスを classpath.txt に追加します。

注意　JAR ファイル用の classpath.txt の必要条件は、ファイル名を指定しない .class ファイルやパッケージに対する条件とは異なります。

たとえば、JAR ファイル e:\java\classes\utilpkg.jar を利用可能にするには、以下を classpath.txt に追加します。

```
e:\java\classes\utilpkg.jar
```

classpath.txt の検索と編集

Java クラスを利用可能にするために、ディレクトリ toolbox/local にあるデフォルトの classpath.txt を編集できます。あるいは、デフォルトの classpath.txt を、このディレクトリからユーザのスタートアップディレクトリにコピーすることができます。この場合、classpath.txt の変更は、他のユーザに影響を与えません。

MATLAB 環境が利用する classpath.txt を検索するには、MATLAB の which コマンドを使います。

```
which classpath.txt
```

デフォルトファイルあるいはユーザディレクトリへのコピーを編集するには、MATLAB でつぎのコマンドを入力します。

```
edit classpath.txt
```

MATLAB は、起動時にのみ classpath.txt を読み込みます。そのため、MATLAB の実行中に classpath.txt を編集したり、.class ファイルを変更する場合は、MATLAB を再起動してこれらの変更を反映させる必要があります。

Java クラス定義を MATLAB にロードする

通常、MATLAB は、コードが最初に Java クラスを用いるときに自動的に Java クラスをロードします (たとえば、コンストラクタのコール時)。しかし、注意すべき例外が 1 つ存在します。

Java クラスにより定義されたメソッドに対して which コマンドを利用するとき、コマンドは MATLAB 作業環境にカレントにロードされているクラスに対してのみ機能します。対照的に、which はそれらがロードされているかどうかにかかわらず、MATLAB クラスに対して常に機能します。

ロードされるクラスの決定

MATLAB セッション中のどんなときでも、カレントにロードされているすべての Java クラスのリストを得ることが可能です。そのためには、つぎの形式で inmem コマンドを使います。

```
[M,X,J] = inmem
```

このコマンドは、出力引数 J に Java クラスのリストを出力します (さらに、カレントでロードされているすべての M- ファイル名を M に、カレントでロードされているすべての MEX- ファイル名を X に出力します)。

以下は、inmem コマンドからの出力の例です。

```
[m,x,j] = inmem
m =
    'isequal'
    'isunix'
    'fullfile'
    'filesep'
    .
    .
    .
    'matlabrc'
x =
    'getprofl'
j =
    'java.awt.Frame'
    'com.mathworks.ide.desktop.MLDesktop'
```

MATLAB での Java クラス名の簡略化

MATLAB コマンドは、パッケージ名を含む修飾子付きの名前によって、Java クラス名を参照できます。たとえば、以下は修飾子付きの名前です。

- java.lang.String
- java.util.Enumeration

修飾子付きの名前は、コンストラクタのように、コマンドや関数を形成していて長いので、編集や読み込みがしにくい場合があります。修飾子付きの名前を MATLAB に最初に import する場合は、(パッケージ名なしで) クラス名のみによってクラスを参照できます。

import コマンドは、次の形式です。

```
import pkg_name.*            % パッケージのクラスを全てインポート
import pkg_name1.* pkg_name2.* % 複数パッケージをインポート
import class_name            % 1 つのクラスをインポート
import                       % カレントインポートリストを表示
L = import                   % カレントインポートリストを返す
```

MATLAB は、import list と呼ばれるリストに import するすべてのクラスを追加します。引数なしで import とタイプすることによって、どのクラスがリストにあるかがわかります。コードは、クラス名のみでリスト上のすべてのクラスを参照できます。

関数から呼ばれたときは、import は指定したクラスを実際にその関数用のインポートリストに追加します。コマンドプロンプトで呼ばれたときは、import は MATLAB 環境に対するベースのインポートリストを利用します。

たとえば、つぎのステートメントを含む関数を考えます。

```
import java.lang.String
import java.util.* java.awt.*
import java.util.Enumeration
```

上記の import ステートメントに続くコードは、パッケージ名を使わずに String, Frame, Enumeration クラスを参照できます。

```
str = String('hello');    % java.lang.String オブジェクトを作成
frm = Frame;              % java.awt.Frame オブジェクトを作成
methods Enumeration       % java.util.Enumeration メソッドを一覧表示
```

インポートされた Java クラスのリストを clear するには、つぎのコマンドを実行します。

```
clear import
```

Java オブジェクトの作成と利用

MATLABでは、そのクラスのコンストラクタの1つを呼び出すことによって、Java オブジェクトを作成します。その後、コマンドとプログラミングステートメントを使って、これらのオブジェクトに対して操作します。Java オブジェクトを MAT-ファイルに保存したり、その後のセッションで MATLAB にリロードすることも可能です。

本節は、以下のトピックスを説明します。

- Java オブジェクトの作成
- Java オブジェクトの結合
- Java オブジェクトの MAT- ファイルへの保存とロード
- オブジェクトのパブリックデータフィールドの検索
- プライベートおよびパブリックデータのアクセス
- オブジェクトのクラスの決定

Java オブジェクトの作成

クラスと同じ名前の Java クラスコンストラクタを呼び出すことによって、Java オブジェクトを MATLAB で作成します。たとえば、つぎのコンストラクタは、タイトルが 'Frame A' であり、他のプロパティはデフォルト値である Frame オブジェクトを作成します。

```
frame = java.awt.Frame('Frame A');
```

新規のオブジェクト frame を表示するのは、以下のステートメントです。

```
frame =
java.awt.Frame[frame0,0,0,0x0,invalid,hidden,layout=
java.awt.BorderLayout,resizable,title=Frame A]
```

本章のすべてのプログラミングの例題は、Java オブジェクトコンストラクタを含みます。たとえば、「URL の読み込み」は、以下のコンストラクタを使って java.net.URL オブジェクトを作成します。

```
url = java.net.URL(...
    'http://www.ncsa.uiuc.edu/demoweb/url-primer.html')
```

javaObject 関数の利用

ある状況においては、javaObject 関数を使って Java オブジェクトを作成する必要があります。つぎのシンタックスは、x1,...,xn に一致する引数リストを使って、クラス class_name 用に Java コンストラクタを呼び出し、新規オブジェクト J を出力します。

```
J = javaObject('class_name',x1,...,xn);
```

たとえば、クラス java.lang.String の Java オブジェクトを作成し、出力するには、以下を使います。

```
strObj = javaObject('java.lang.String','hello');
```

javaObject 関数を使って、以下のことが可能です。

- 31 文字よりも長い名前をもつクラスを利用。
- たとえば、アプリケーションのユーザからの入力として、実行時にオブジェクトに対するクラスを指定。

デフォルトの MATLAB コンストラクタのシンタックスは、入力クラス名のセグメントが 31 文字より長くないことが必要です (クラス名セグメントは、ピリオドの前、間、後のクラス名の部分です。たとえば、クラス java.lang.String には、3 つのセグメントがあります)。31 文字を超えるすべてのクラス名セグメントは、MATLAB により打ち切られます。まれな場合に、この長さのクラス名を使う必要があるときは、javaObject を使ってクラスをインスタンスしなければなりません。

javaObject 関数を使って、実行時に作成されるオブジェクトに対する Java クラスを指定することも可能です。この状況において、クラス名の引数の代わりに文字列変数を使って、javaObject を呼び出します。

```
class  = 'java.lang.String';
text = 'hello';
strObj = javaObject(class, text);
```

通常の場合、インスタンスするクラスが開発時にわかっているときは、MATLAB コンストラクタのシンタックスを使うとより便利です。たとえば、java.lang.String オブジェクトを作成するには、以下を使います。

```
strObj = java.lang.String('hello');
```

注意　一般に、javaObject を使う必要はありません。Java クラスをインスタンスするデフォルトの MATLAB シンタックスは簡単で、ほとんどのアプリケーションに対して望ましいものです。主として、上記に説明した 2 つの場合に対して javaObject を使ってください。

Java オブジェクトは、MATLAB のリファレンスです

MATLAB では、Java オブジェクトはリファレンスであり、MATLAB の copy-on-assignment および pass-by-value のルールに従いません。たとえば、

```
origFrame = java.awt.Frame;
setSize(origFrame, 800, 400);
newFrameRef = origFrame;
```

上記の 2 番目のステートメントで、変数 newFrameRef は、origFrame の 2 番目のリファレンスであり、オブジェクトのコピーではありません。上記の例のつぎのコードにおいて、newFrameRef でのオブジェクトの変更は、origFrame のオブジェクトを変更します。この影響は、オブジェクトが MATLAB コードによって、あるいは Java コードによって変更されるかどうかにより発生します。

つぎの例は、origFrame と newFrameRef が両方共同じ要素のリファレンスであることを示します。フレームのサイズがあるリファレンス (newFrameRef) によって変更されるとき、変更はもう 1 つのリファレンス (origFrame) に反映されます。

```
setSize(newFrameRef, 1000, 800);

getSize(origFrame)
ans =
java.awt.Dimension[width=1000,height=800]
```

Java オブジェクトの結合

ネイティブな MATLAB データタイプを結合するのと同じ方法で Java オブジェクトを結合することができます。cat コマンドあるいは中括弧を使って、囲まれたオブジェクトを 1 つのオブジェクトにまとめることを MATLAB に通知します。操作されるすべてのオブジェクトが同じ Java クラスである場合は、これらのオブジェクトの結合は、同じクラスのオブジェクトの配列になります。

つぎの例で、cat コマンドは、クラス java.awt.Point の 2 つのオブジェクトを結合します。結果のクラスも、java.awt.Point です。

注意　cat コマンドを使うとき、第一引数 (結合を行う軸を指定) は 1 でなければなりません。Java オブジェクトは、第一の軸に沿ってのみ結合されます。

```
point1=java.awt.Point(24,127);
point2=java.awt.Point(114,29);

cat(1, point1, point2)
ans =
java.awt.Point[]:
   [1x1 java.awt.Point]
   [1x1 java.awt.Point]
```

2 番目の例で、MATLAB の中括弧演算子は、異なるクラスの 2 つのオブジェクトを結合するために用いられます。結果のクラスは、Java クラス階層のルートである java.lang.Object です。

```
frame=java.awt.Frame('Sample Frame');

[frame point1]
ans =
java.lang.Object[]:
   [1x1 java.awt.Frame]
   [1x1 java.awt.Point]
```

Java オブジェクトの MAT- ファイルへの保存とロード

MATLABのsaveコマンドを使って、Javaオブジェクトを MAT-ファイルに保存します。load コマンドを使って MAT- ファイルから MATLAB へとロードします。Java オブジェクトを MAT-ファイルに保存し、MAT-ファイルからオブジェクトをロードするには、オブジェクトとクラスがつぎの規範のすべてを満たすことを確認してください。

- クラスは、直接あるいは、親のクラスから継承することによって、Serializable インタフェース (Java API の一部) を実現します。組み込まれたオブジェクトあるいは参照されたオブジェクトも Serializable を実装する必要があります。
- クラスの定義は、オブジェクトの保存とロードの間で変更されません。データフィールド、あるいはクラスのメソッドへの変更は、古いクラス定義によって作成されたオブジェクトのロード (deserialization) を防ぎます。

- クラスは、一時的データフィールドをもたないか、あるいは保存されるオブジェクトの一時的データフィールドの値は、意味がありません。一次的データフィールドのデータは、オブジェクトと共に保存されません。

ユーザ独自の Java クラスか、既存のクラスのサブクラスを定義する場合は、クラスのオブジェクトを MATLAB で保存およびロード可能にするために上記の規範に従います。serialization をサポートするクラスの定義に関する詳細は、Java 開発のドキュメントを参照してください（「Java プログラミングについて学習する」も参照してください）。

オブジェクトのパブリックデータフィールドの検索

Java オブジェクトに属するパブリックフィールドをリストするには、つぎのいずれかの形式の fieldnames 関数を使います。

```
names = fieldnames(obj)
names = fieldnames(obj,'-full')
```

'-full'を付けずにfieldnamesを呼び出すと、オブジェクトのすべてのデータフィールド（継承を含む）名を出力します。'-full' 修飾子を付けると、fieldnames は、タイプ、属性、継承の情報を含むオブジェクトに対して定義されたデータフィールドの詳しい説明を出力します。

たとえば、以下を使って Frame オブジェクトを作成したとします。

```
frame = java.awt.Frame;
```

frame のデータフィールドの詳しい情報を得るには、以下のコマンドを使います。

```
fieldnames(frame,'-full')
```

このコマンドの出力例は、以下の通りです。

```
ans =
'static final int WIDTH
   % java.awt.image.ImageObserver' から継承
'static final int HEIGHT
   % java.awt.image.ImageObserver' から継承
[1x74 char]
'static final int SOMEBITS
   %  java.awt.image.ImageObserver' から継承
'static final int FRAMEBITS
   %  java.awt.image.ImageObserver' から継承
'static final int ALLBITS
```

```
    % java.awt.image.ImageObserver' から継承
  'static final int ERROR
    % java.awt.image.ImageObserver' から継承
  'static final int ABORT
    % java.awt.image.ImageObserver' から継承
  'static final float TOP_ALIGNMENT
    %  java.awt.Component' から継承
  'static final float CENTER_ALIGNMENT
    %  java.awt.Component' から継承
  'static final float BOTTOM_ALIGNMENT
    % java.awt.Component' から継承
  'static final float LEFT_ALIGNMENT
    %java.awt.Component' から継承
  'static final float RIGHT_ALIGNMENT
    %  java.awt.Component' から継承
 .
 .
 .
```

プライベートおよびパブリックデータのアクセス

Java API クラスは、ユーザが読み込んだり、可能な場合は*プライベート*データフィールドを修正するために利用できるアクセサメソッドを提供します。これらは、それぞれ get および set メソッドとして参照されることがあります。

Java クラスの中には、*パブリック*データフィールドをもつものがあります。これは、ユーザコードが直接読み込んだり、修正したりできます。これらのフィールドにアクセスするには、シンタックス object.field を使います。

例題

java.awt.Frame クラスは、プライベートとパブリックの両方のデータフィールドにアクセスする例を提供しています。このクラスは、読み込みのアクセサメソッド getSize を提供し、これは java.awt.Dimension オブジェクトを出力します。Dimension オブジェクトは、パブリックで直接アクセスが可能なデータフィールド height および width をもちます。つぎの例は、このデータにアクセスする MATLAB コマンドを示します。

```
frame = java.awt.Frame;
frameDim = getSize(frame);
height = frameDim.height;
frameDim.width = 42;
```

本章のプログラミングの例題は、データフィールドのアクセサの呼び出しも含みます。たとえば、「インターネットプロトコルアドレスの検索」のサンプルコードは、java.net.InetAddress オブジェクトのアクセサの呼び出しを使います。

```
hostname = address.getHostName;
ipaddress = address.getHostAddress;
```

スタティックフィールドからデータにアクセスする

Java では、スタティックデータフィールドは、オブジェクトのクラス全体に適用するフィールドです。スタティックフィールドは、Java でのクラス名に関連して最も一般にアクセスされます。たとえば、下記のコードは、オブジェクトのインスタンスでなく、パッケージとクラス名 java.awt.Frame に関して参照することにより、Frame クラスの WIDTH フィールドにアクセスします。

```
width = java.awt.Frame.WIDTH;
```

MATLAB では、同じシンタックスを使うことができます。あるいは、クラスのインスタンスに関して WIDTH フィールドを参照することができます。この例題は、frameObj と呼ばれる java.awt.Frame のインスタンスを作成し、パッケージとクラス名ではなく名前 frameObj を使って、WIDTH フィールドにアクセスします。

```
frame = java.awt.Frame('Frame A');

width = frame.WIDTH
width =
   1
```

スタティックフィールドの割り当て

クラスのスタティックな set メソッドを使うか、あるいはクラスのインスタンスのリファレンスに割り当てを行うことにより、スタティックな Java フィールドに値を割り当てることができます。詳細は、前の節 " スタティックフィールドからデータにアクセスする " を参照してください。クラスのインスタンスのリファレンスにおいてこのフィールドを参照することによって、フィールド staticFieldName に値を割り当てることができます。

```
objectName = java.className;
objectName.staticFieldName = value;
```

注意　MATLAB は、クラス名自身を使ってスタティックなフィールドを割り当てできません。

オブジェクトのクラスの決定

Java オブジェクトのクラスを決定するには、MATLAB 関数 class のクエリー形式を使います。つぎの例題の実行の後で、frameClass には、Java オブジェクト frame をインスタンスするパッケージとクラス名が含まれます。

```
frameClass = class(frame)
frameClass =
java.awt.Frame
```

この形式の class は MATLAB オブジェクトに対しても機能するため、それ自身では Java クラスであるかどうかを判断できません。クラスのタイプを決定するには、つぎの形式の isjava 関数を使います。

```
x = isjava(obj)
```

obj が Java ならば isjava は 1 を出力し、そうでない場合は 0 を出力します。

```
isjava(frame)
ans =
   1
```

オブジェクトが指定したクラスのインスタンスであるかどうかを求めるためには、つぎの形式の isa 関数を使います。

```
x = isa(obj, 'class_name')
```

obj が 'class_name' という名前のクラスのインスタンスである場合は isa は 1 を出力し、そうでない場合は 0 を出力します。'class_name' は、Java クラスと同様に、MATLAB の組み込みのクラスあるいはユーザ定義クラスでもかまいません。

```
isa(frame, 'java.awt.Frame')
ans =
   1
```

Java オブジェクトのメソッドの実行

本節は、MATLAB でのオブジェクトのメソッドの実行方法を説明します。また、利用するメソッドに関する情報の取得方法や MATLAB がどのように非標準の状況のタイプを扱うかについても説明します。

本節は、以下のトピックスを説明します。

- Java および MATLAB の呼び出しのシンタックスの使用法
- Java クラスにおけるスタティックメソッドの呼び出し
- メソッドに関する情報の取得
- MATLAB コマンドに影響する Java メソッド
- MATLAB が未定義のメソッドをどのように取り扱うか
- MATLAB が Java の例外をどのように取り扱うか

Java および MATLAB の呼び出しのシンタックスの使用法

Java オブジェクトにおいてメソッドを呼び出すには、Java のシンタックスを利用します。

```
object.method(arg1,...,argn)
```

つぎの例では、frame は上記で作成された java.awt.Frame オブジェクトで、getTitle と setTitle はそのオブジェクトのメソッドです。

```
frame.setTitle('Sample Frame')

title = frame.getTitle
title =
Sample Frame
```

あるいは、MATLAB のシンタックスを使って Java オブジェクト（非スタティック）メソッドを呼び出すことができます。

```
method(object, arg1,...,argn)
```

MATLAB シンタックスでは、上記の java.awt.Frame の例題はつぎのようになります。

```
setTitle(frame, 'Sample Frame')
```

```
title = getTitle(frame)
title =
Sample Frame
```

本章のすべてのプログラミングの例題は、Java オブジェクトメソッドの呼び出しを含みます。たとえば、「URL の読み込み」のコードは、MATLAB シンタックスを使った java.net.URL オブジェクト url に対する openStream メソッドの呼び出しを含みます。

```
is = openStream(url)
```

他の例題では、「Phone Book の作成と利用」のコードは、Java シンタックスを使った、java.utils.Properties オブジェクト pb_htable に対する load メソッドの呼び出しを含みます。

```
pb_htable.load(FIS);
```

非スタティックメソッドに対して javaMethod 関数を利用

ある状況下では、javaMethod 関数を使って Java メソッドを呼び出す必要がある場合があります。つぎのシンタックスは、x1,...,xn に一致する引数リストを使って、method_name, メソッドを Java オブジェクト J について呼び出します。これは、値 X を出力します。

```
X = javaMethod('method_name',J,x1,...,xn);
```

たとえば、1 つの引数を渡して java.lang.String オブジェクトについて startsWith メソッドを呼び出すには、以下を使います。

```
gAddress = java.lang.String('Four score and seven years ago');
str = java.lang.String('Four score');

javaMethod('startsWith', gAddress, str)
ans =
    1
```

javaMethod 関数を使って、以下のことが行えます。

- 31 文字よりも長い関数名を持つメソッドを使うことができます。
- たとえば、アプリケーションユーザからの入力として実行時に呼び込みたいメソッドを指定できます。

31 文字よりも長い名前を持つメソッドを呼び出す唯一の方法は、javaMethod を使うことです。「Java と MATLAB の呼び出しシンタックス」で説明されている Java と MATLAB の呼び出しシンタックスは、この長さのメソッド名を扱えません。

javaMethod を使って、実行時に呼び出されるメソッドを指定することもできます。この状況で、ユーザコードは、メソッド名引数の代わりに文字列変数を使って javaMethod を呼び出します。javaMethod を使ってスタティックメソッドを呼び出すとき、クラス名引数の代わりに文字列変数を使うこともできます。

注意　一般的に、javaMethod を使う必要はありません。Java メソッドの呼び出しに対するデフォルトの MATLAB シンタックスは、簡単で、ほとんどのアプリケーションに対して望ましいものです。主として、上記で説明した 2 つの場合に対して javaMethod を使ってください。

Java クラスに対するスタティックメソッドの呼び出し

Java クラスに対してスタティックメソッドを呼び出すには、Java の呼び出しシンタックスを使います。

```
class.method(arg1,...,argn)
```

たとえば、java.lang.Double クラスに対して isNaN スタティックメソッドを呼び出します。

```
java.lang.Double.isNaN(2.2)
```

あるいは、クラスのインスタンスにスタティックなメソッド名を適用できます。この例題では、isNaN スタティックメソッドは、java.lang.Double クラスの dblObject インスタンスに関して参照されます。

```
dblObject = java.lang.Double(2.2);
dblObject.isNaN
ans =
    0
```

本章のプログラミングの例題の中には、スタティックなメソッドの呼び出しの例を含むものがあります。たとえば、シリアルポートによる通信のためのコードは、javax.comm.CommPortIdentifierに対するスタティックメソッドgetPortIdentifierの呼び出しを含みます。

```
commPort = javax.comm.CommPortIdentifier.getPortIdentifier('COM1');
```

スタティックメソッドに対して javaMethod 関数を利用

javaMethod 関数は、「非スタティックメソッドに対して javaMethod 関数を利用」で説明されています。この関数を使ってスタティックメソッドを呼び出すこともできます

つぎのシンタックスは、x1,...,xn に一致する引数リストを使ってクラス class_name のスタティックメソッドmethod_nameを呼び出します。これは、値Xを出力します。

```
X = javaMethod('method_name','class_name',x1,...,xn);
```

たとえば、2.2 の double の値に対して java.lang.Double クラスのスタティックメソッド isNaN を呼び出すには、以下を使います。

```
javaMethod('isNaN','java.lang.Double',2.2);
```

javaMethod 関数を使ってスタティックメソッドを呼び出すと、以下のことが行えます。

- 31 文字よりも長い名前を持つメソッドを利用できます。
- たとえば、アプリケーションユーザからの入力として、実行時にメソッドとクラス名を指定できます。

メソッドに関する情報の取得

MATLAB は、作業を行う Java メソッドに関連する情報を取得するためのコマンドを提供しています。クラスが実現しているすべてのメソッドの一覧を要求することができます。リストは、引数のタイプや例外のようなその他のメソッドの情報と共に提供されている場合があります。指定したメソッドを実現する MATLAB にロードされた Java クラスのリストを要求することも可能です。

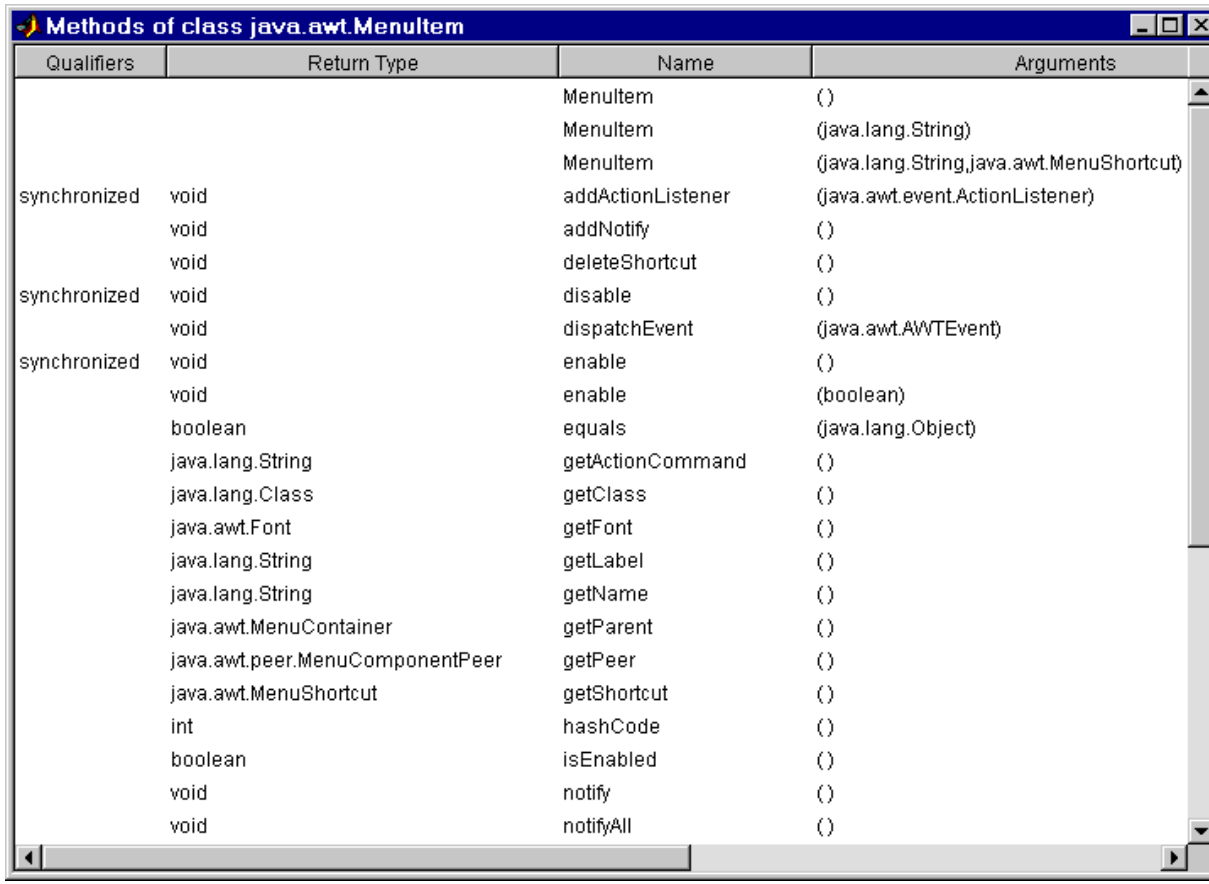
Methodsview: Java メソッドのリストの表示

特定の Java(あるいは MATLAB) クラスによってどのメソッドが実現されるかを知りたい場合は、MATLAB で methodsview コマンドを使います。コマンドラインで (Java クラスに対するパッケージ名と共に) クラス名を指定します。このクラスを定義するパッケージをインポートした場合、クラス名のみで十分です。

つぎのコマンドは、java.awt.MenuItem クラスの全てのメソッドに関する情報をリストします。

```
methodsview java.awt.MenuItem
```

クラス内の各メソッドに対して 1 行の情報を表示する新規のウィンドウが現れます。以下は、methodsview の表示です。ウィンドウの一番上のフィールド名は、図のつぎに示します。



Qualifiers	Return Type	Name	Arguments
		MenuItem	()
		MenuItem	(java.lang.String)
		MenuItem	(java.lang.String,java.awt.MenuShortcut)
synchronized	void	addActionListener	(java.awt.event.ActionListener)
	void	addNotify	()
	void	deleteShortcut	()
synchronized	void	disable	()
	void	dispatchEvent	(java.awt.AWTEvent)
synchronized	void	enable	()
	void	enable	(boolean)
	boolean	equals	(java.lang.Object)
	java.lang.String	getActionCommand	()
	java.lang.Class	getClass	()
	java.awt.Font	getFont	()
	java.lang.String	getLabel	()
	java.lang.String	getName	()
	java.awt.MenuContainer	getParent	()
	java.awt.peer.MenuComponentPeer	getPeer	()
	java.awt.MenuShortcut	getShortcut	()
	int	hashCode	()
	boolean	isEnabled	()
	void	notify	()
	void	notifyAll	()

ウィンドウ内の各行は、メソッドを説明する最大 6 フィールドの情報を表示します。下記の表は、各フィールドタイプの説明や例と共に methodsview ウィンドウに表示されるフィールドの一覧です。

表 5-1：Methodsview ウィンドウに表示されるフィールド

フィールド名	説明	例
Qualifiers	メソッドタイプ修飾子	abstract, synchronized
Return Type	メソッドにより出力されるデータタイプ	void, java.lang.String
Name	メソッド名	addActionListener, dispatchEvent
Arguments	メソッドに渡される引数	boolean, java.lang.Object
Other	その他の関連情報	throws java.io.IOException
Parent	指定したクラスの親	java.awt.MenuComponent

Java クラスに対する Methods コマンドの利用

methodsview に加えて、MATLAB クラスのメソッドに関する情報を出力する MATLAB の methods コマンドも、Java クラスに対して機能します。つぎのいずれかの形式でコマンドを使うことができます。

```
methods class_name
methods class_name -full
n = methods('class_name')
n = methods('class_name','-full')
```

'-full'修飾子を付けずにmethodsを呼び出すと、(継承されたメソッドを含む)クラスのすべてのメソッド名を出力します。オーバロードされるメソッド名は 1 回だけリストされます。

'-full' 修飾子を付けると、methods は、それぞれの属性、引数リスト、継承の情報と共に (継承されたメソッドを含む) メソッド名のリストを出力します。オーバロードされたメソッドは、別にリストされます。

たとえば、java.awt.Dimension object オブジェクトのすべてのメソッドの詳しい説明を表示します。

```
methods java.awt.Dimension -full

Methods for class java.awt.Dimension:
Dimension()
Dimension(java.awt.Dimension)
Dimension(int,int)
java.lang.Class getClass()  % java.lang.Object から継承
int hashCode()  % java.lang.Object から継承
boolean equals(java.lang.Object)
java.lang.String toString()
void notify()  % java.lang.Object から継承
void notifyAll()  %  java.lang.Object から継承
void wait(long) throws java.lang.InterruptedException
  % java.lang.Object から継承
void wait(long,int) throws java.lang.InterruptedException
  %  java.lang.Object から継承
void wait() throws java.lang.InterruptedException
  % java.lang.Object から継承
java.awt.Dimension getSize()
void setSize(java.awt.Dimension)
void setSize(int,int)
```

どのクラスがメソッドを定義するかを決定

whichコマンドを使って、ロード されたJavaクラスにより実現されたメソッドの (パッケージとクラス名) 修飾子付きの名前を表示することができます。-all 修飾子を付けると、which コマンドは、指定された名前のメソッドを持つすべてのクラスを検索します。

たとえば、Stringクラスが現在ロードされているconcatメソッドに対するパッケージとクラス名を探したいとします。つぎのコマンドを使います。

```
which concat
java.lang.String.concat  % String メソッド
```

java.lang.String クラスがロードされていない場合は、同じ which コマンドは、つぎの出力を与えます。

```
which concat
concat not found.
```

String および java.awt.Frame クラスがロードされているメソッド equals に対して which -all を使う場合は、以下が表示されます。

```
which -all equals
java.lang.String.equals                 % String メソッド
java.awt.Frame.equals                   % Frame メソッド
com.mathworks.ide.desktop.MLDesktop.equals  % MLDesktop メソッド
```

which コマンドは、MATLAB クラスと Java クラスに対しては異なって機能します。MATLAB クラスは、それらがロードされていてもいなくても、which によって常に表示されます。これは、Java クラスに対してはあてはまりません。「どのクラスがロードされているかを決定する」に説明されているコマンド [m,x,j]=inmem を使って、どの Java クラスが現在ロードされているかを知ることができます。

Java クラスがどのようにロードされるかに関する説明は、「MATLAB で Java クラスを利用可能にする」を参照してください。

MATLAB コマンドに影響を与える Java メソッド

Java オブジェクトおよび配列に対して機能する MATLAB コマンドは、これらのオブジェクトのクラスによって実現されるか、あるいは継承されるメソッドを利用します。MATLAB コマンドの中には、依存する Java メソッドを変更することによって、挙動を多少変更することができるものがあります。

DISP および DISPLAY の効果の変更

disp コマンドを使って MATLAB において変数や式の値を表示することができます。セミコロンを使わずにコマンドラインを終了させても、disp 関数を呼び出します。disp を使って MATLAB に Java オブジェクトを表示することも可能です。

disp が Java オブジェクトに対して機能するとき、MATLAB は、オブジェクトが属するクラスの toString メソッドを使って、出力を書式化します。クラスがこのメソッドを実現しない場合は、継承された toString メソッドが用いられます。中間の ancestor クラスがこのメソッドを定義しない場合、java.lang.Object クラスによって定義された toString メソッドを用います。そのようなメソッドをクラス定義内で実現することによって、作成するクラス内の継承された toString メソッドを変更することができます。この方法で、MATLAB がクラスのオブジェクトに関する情報を表示する方法を変更することができます。

ISEQUAL の効果の変更

MATLAB の isequal コマンドは、2 つあるいはそれ以上の配列のタイプ、サイズ、内容の等しさを比較します。このコマンドは、Java オブジェクトの等しさをテストするためにも用いられます。

isequalを使って2つのJavaオブジェクトを比較するとき、MATLABはJavaメソッド equals を使って比較を行います。MATLAB は、最初にコマンドで指定されたオブジェクトのクラスを決定し、つぎにクラスによって実現された equals メソッドを使います。このクラス内で実現されていない場合は、継承された equals メソッドが用いられます。中間の ancestor クラスがこのメソッドを定義しない場合は、これは、java.lang.Object クラスによって定義された equals メソッドです。

クラス定義内でそのようなメソッドを実現することによって、作成したクラス内の継承された equals メソッドを変更することができます。この方法で、MATLAB がクラスのメンバの比較を行う方法を変更することができます。

DOUBLE および CHAR の効果の変更

JavaメソッドtoDoubleおよびtoCharを定義して、MATLABのdoubleおよびcharコマンドの出力を変更することも可能です。詳細は、「MATLAB double データタイプへの変換」と「MATLAB char データタイプへの変換」を参照してください。

未定義のメソッドの取り扱い方法

MATLABコマンドがJavaオブジェクトに対して存在しないメソッドを呼び出す場合、MATLAB は同じ名前の組み込み関数を探します。MATLAB が同名の組み込み関数を見つけた場合、それを呼び出そうとします。MATLAB がその名前の関数を見つけられない場合、クラスに対してその名前のメソッドが見つからないことを示すメッセージを表示します。

たとえば、MATLAB は、size という名前の組み込みのメソッドを持ち、Java API java.awt.Frameクラスもsizeメソッドを持ちます。Frameオブジェクトに対してsizeを呼び出す場合、java.awt.Frame によって定義された size メソッドが実行されます。しかし、java.lang.String のオブジェクトに対して size を呼び出す場合は、MATLAB はこのクラスに対する size メソッドを見つけません。その代わりにMATLABの組み込みsizeを実行します。以下のサンプルコードを参照してください。

```
string = java.lang.String('hello');

size(string)
ans =
```

1 1

注意 MATLAB での利用のために Java クラスを定義するとき、メソッドに MATLAB 組み込み関数と同じ名前を付けることは避けてください。

Java の例外の取り扱い方法

Java メソッド、または、コンストラクタの呼び出しが例外を与える場合は、MATLAB は例外をとらえて、MATLAB エラーに変換します。MATLAB は Java エラーメッセージのテキストをエラーメッセージに変換します。Java メソッドあるいはコンストラクタからエラーを受信すると、M- ファイルからエラーを受信したのと同じ表示を行います。

Java 配列の機能

Java オブジェクトをメソッドに渡したり、メソッドがその形式であると予想される場合には、配列に渡すことができます。この配列は、Java 配列 (他のメソッド呼び出しから出力、あるいは MATLAB 内で作成される)、あるいはある状況下では MATLAB セル配列でなければなりません。本節では、MATLAB での Java 配列の作成と操作方法を説明します。後の節では、Java メソッドの呼び出しでの MATLAB セル配列の利用法を説明します。

注意 ここで、用語 dimension は、配列の長さ、幅、高さの特性ではなく、配列の要素を示すために必要なサブスクリプトの数を表わします。たとえば、5 行 1 列配列は、個々の要素が 1 つの配列のサブスクリプトのみを使ってインデックス付けられている 1 次元であるとして参照されます。

以下のトピックスを説明します。

- Java 配列の表現方法
- MATLAB 内部でのオブジェクトの配列の作成
- Java 配列の要素へのアクセス
- Java 配列の割り当て
- Java 配列の結合
- 新規の配列の参照の作成
- Java 配列のコピーの作成

Java 配列の表現方法

用語「*java 配列*」は、Java クラスコンストラクタまたはメソッドの呼び出しから出力された Java オブジェクトの配列を表わします。javaArray 関数を使ってMATLAB内部でJava配列を作成することも可能です。Java配列の構造は、MATLAB行列や配列の構造とは明らかに異なります。MATLAB は、通常の MATLAB のコマンドシンタックスを使って配列を操作できるように、可能な限りこれらの差異を隠します。同様に、Java 配列を操作するときにつぎの違いを注意することが役立つ場合があります。

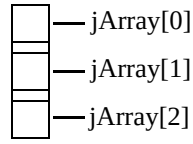
2 次元以上の表現

Java 言語における配列は、厳密に 1 次元構造で、長さでのみ判断されます。2 次元配列を操作したい場合は、配列の配列を使って同じ構造を作成することができます。さらに次元を追加するには、配列の配列の配列として配列のレベルを追加します。行列や配列ベースのプログラミング言語として MATLAB で操作するときには、そのようなマルチレベルの配列を使いたい場合があります。

MATLAB は、言語自体の一部分である行列や多次元配列のように取り扱うことにより、マルチレベルの Java 配列の操作を簡単にします。行列を取り扱う場合に利用するのと同じ MATLAB シンタックスを使って、配列の配列の要素にアクセスします。配列にさらにレベルを追加する場合は、MATLAB は、多次元 MATLAB 配列であるかのように構造体にアクセスし、操作することが可能です。

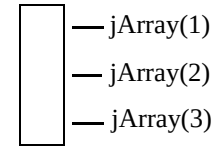
下図の左側は、1 次元、2 次元、3 次元の Java 配列を表わします。各々の右側は、MATLABで同じ配列が表わされる方法です (1次元配列は、列ベクトルとして表わされます)。

Array Access from Java

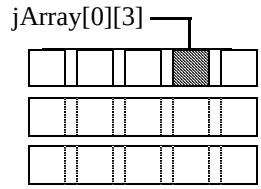


Simple Array

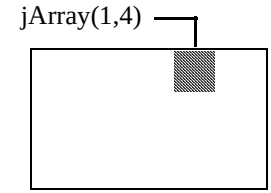
Array Access from MATLAB



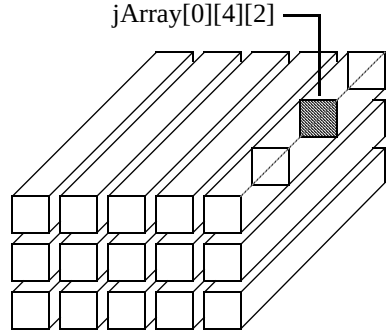
One-dimensional Array



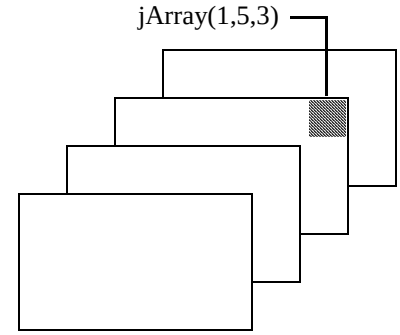
Array of Arrays



Two-Dimensional Array



Array of Arrays of Arrays



Three-Dimensional Array

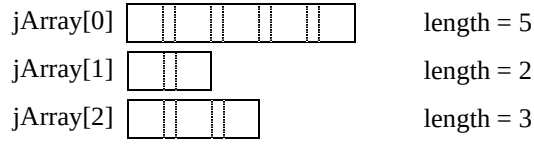
配列のインデックス

Java 配列のインデックスは、MATLAB 配列のインデックスとは異なります。Java 配列のインデックスはゼロベースで、MATLAB 配列のインデックスは 1 ベースです。Java プログラミングでは、y[0] から y[N-1] を使って長さ N の配列 y の要素にアクセスします。MATLAB でこれらの配列を操作するときは、y(1) から y(N) の MATLAB のインデックスのスタイルを使って同じ要素にアクセスします。そのため、10 要素の Java 配列があるとき、7 番目の要素は、Java でプログラムを書くときに用いる y[6] ではなく、y(7) を使って得られます。

Java 配列の形状

Java 配列は、全体の*形状* が MATLAB 配列とは異なる場合があります。2 次元の MATLAB 配列は、各行が同じ長さで、各列が同じ高さである長方形の形状です。この配列の Java 版は、配列の配列で、この長方形形状を必ずしも保持しません。個々の低レベルの配列は、長さが異なる場合があります。

そのような配列の構造は、下記のように表わされます。これは、異なる長さの 3 つの基底の配列の配列です。用語 *ragged* は、配列の終端が一様でない配列の要素の配置を記述するために一般に用いられます。Java メソッドがこのタイプの構造の配列を出力するとき、MATLAB によりセル配列に格納されます。

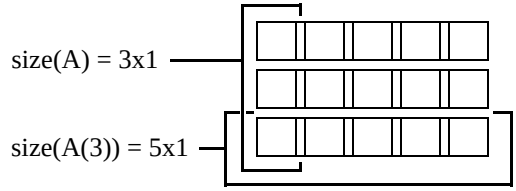


Java 配列のサイズの解釈

MATLABのsizeコマンドを簡単なJava配列に適用するとき、出力される行数はJava配列の長さで、出力される列数は常に 1 です。

Java 配列の配列のサイズを決定することは容易ではありません。Java によって出力される、潜在的にでこぼこな形状である配列は、長方形行列と同じ方法で配列のサイズを決定することが不可能です。でこぼこの Java 配列では、低レベル配列のサイズを表わす単一の値はありません。

size コマンドを Java 配列の配列に適用するとき、結果の値は指定した配列の最上位を表わします。下記の Java 配列に対して、



size(A) は、A の最上位の配列の次元を出力します。配列の最上位は、サイズ 3 行 1 列です。

```
size(A)
ans =
   3    1
```

低レベルの配列のサイズ、たとえば行 3 の 5 要素配列を求めるには、行を明示的に指定します。

```
size(A(3))
ans =
   5    1
```

つぎのシンタックスを使って、size コマンドで次元を指定できます。しかし、これは非単項の次元のみであるため、最初の次元 (dim=1) のサイズのみに対して有効です。

```
m = size(X,dim)
```

```
size(A, 1)
ans =
   3
```

Java 配列の次元数の解釈

Java 配列に対して、単純な 1 レベルの配列であるか、マルチレベルであっても、MATLAB の ndims コマンドは常に値 2 を出力し、配列内の次元数を表わします。これは、常に 2 に等しい最上位配列の次元数の基準です。

MATLAB 内でのオブジェクトの配列の作成

Java オブジェクトの配列として定義された 1 つまたは複数の引数をもつ Java メソッドを呼び出すには、ほとんどの場合、Java 配列内のオブジェクトを渡す必要があります。Java メソッドあるいはコンストラクタの呼び出しにおいてオブジェクトの配列を作成できます。あるいは、MATLAB 内部で配列を作成できます。

MATLABのjavaArray関数を使って、MATLABで単一の多次元配列として扱われるJava 配列構造体を作成します。配列の次元数とサイズと共に、それらを格納するオブジェクトのクラスを指定します。1 次元の Java 配列を基本の作成ブロックとして利用して、MATLAB は javaArray コマンドで要求される次元を満たす配列構造を作成します。

javaArray 関数の使用法

Java オブジェクト配列を作成するには、MATLAB の javaArray 関数を使います。これはつぎのシンタックスです。

```
A = javaArray('element_class', m, n, p, ...)
```

第一引数は、'element_class' 文字列で、配列内の要素のクラス名です。修飾子付きの名前 (パッケージとクラス名) を指定しなければなりません。その他の引数 (m, n, p, ...) は、配列の各次元内の要素数です。

javaArray を使って作成した配列は、以下の Java コードを使って作成した配列と同じです。

```
A = new element_class[m][n][p]...;
```

つぎのコマンドは、各々が java.lang.Double クラスの 5 個のオブジェクトをもつ 4 つの低レベルの配列の Java 配列を作成します (おそらく java.lang.Double クラスのインスタンスよりも基本的な double タイプを使うかもしれませんが、このコンテキストでは簡単な例題を与えます)。

```
dblArray = javaArray('java.lang.Double', 4, 5);
```

javaArray 関数は、作成する配列要素に値を置きません。値は別個に置かなければなりません。つぎの MATLAB コードは、作成した Java 配列 dblArray に java.lang.Double タイプのオブジェクトを格納します。

```
for i = 1:4
   for j = 1:5
   dblArray(i,j) = java.lang.Double((i*10) + j);
   end
end

dblArray
dblArray =
java.lang.Double[][]:
   [11]   [12]   [13]   [14]   [15]
   [21]   [22]   [23]   [24]   [25]
   [31]   [32]   [33]   [34]   [35]
   [41]   [42]   [43]   [44]   [45]
```

Java 配列を作成するその他の方法

MATLABの典型的なシンタックスを使ってjavaオブジェクトの配列を作成することも可能です。たとえば、つぎのシンタックスは、タイプ double の 4 行 5 列の MATLAB 配列を作成し、配列の各要素にゼロを割り当てます。

```
matlabArray(4,5) = 0;
```

Java クラス名を指定する必要があること以外は、同じシンタックスを使ってMATLABでJava配列を作成します。この場合では0に割り当てられた値は、配列の最後の要素 javaArray(4,5) に格納されます。配列のその他のすべての要素は、空行列を受け取ります。

```
javaArray(4,5) = java.lang.Double(0)
javaArray =
java.lang.Double[][]:
   []   []   []   []   []
   []   []   []   []   []
   []   []   []   []   []
   []   []   []   []   [0]
```

注意　ユーザが、既に存在している Java 配列の次元を、MATLAB 配列で行うように変更することはできません。Java 言語の中で、Java 配列を取り扱う場合、同じ制約が存在します。つぎの例題を参照してください。

つぎの例題は、まず、スカラの MATLAB 配列を作成し、その後で、2 次元にそれを変更するものです。

```
matlabArray = 0;
matlabArray(4,5) = 0

matlabArray =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Java 配列を使って、これを行う場合、エラーが生じます。同様に、Java 配列から複数の Java 配列を作成することはできません。

```
javaArray = java.lang.Double(0);
javaArray(4,5) = java.lang.Double(0);
??? Index exceeds Java array dimensions.
```

Java 配列の要素へのアクセス

MATLAB の配列のインデックス付けシンタックス A(row,col) を使って、Java オブジェクト配列の要素にアクセスできます。たとえば、行 3、列 4 の配列 dblArray の要素にアクセスするには、以下を使います。

```
row3_col4 = dblArray(3,4)
row3_col4 =
34.0
```

Java では、これは dblArray[2][3] です。

MATLAB の配列のインデックス付けのシンタックスを使って、オブジェクトのデータフィールドの要素にアクセスすることも可能です。myMenuObj が window menu クラスのインスタンスであるとします。このユーザ定義クラスは、java.awt.menuItem の Java 配列であるデータフィールド menuItemArray をもちます。この配列の要素 3 を取得するには、つぎのコマンドを使います。

```
currentItem = myMenuObj.menuItemArray(3)
```

単一のサブスクリプトのインデックスを使った配列のアクセス

MATLAB 行列の要素は、行と列のサブスクリプトを使って最も一般的に参照されます。たとえば、行 3、列 4 の交点にある配列要素を参照するには、x(3,4) を使

います。単一のサブスクリプトを使うとより有利である場合があります。MATLAB は、この機能を提供しています (「アドバンスドなインデックス」を参照)。

単一のサブスクリプトを使った MATLAB 行列のインデックス付けは、行列の 1 要素を参照します。下記の MATLAB 行列を使って、matlabArray(3) は行列の 1 要素を出力します。

```
matlabArray = [11 12 13 14 15; 21 22 23 24 25; ...
          31 32 33 34 35; 41 42 43 44 45]
matlabArray =
   11   12   13   14   15
   21   22   23   24   25
   31   32   33   34   35
   41   42   43   44   45

matlabArray(3)
ans =
   31
```

この方法で配列の Java 配列にインデックスを付けると、構造体全体のサブ配列を参照します。上記の matlabArray と同じように見える dblArray Java 配列を使うと、dblArray(3) は、3 行目を構成する 5 行 1 列配列を出力します。

```
row3 = dblArray(3)
row3 =
java.lang.Double[]:
   [31]
   [32]
   [33]
   [34]
   [35]
```

これは、大きい配列構造から配列全体を指定することができ、それをオブジェクトとして操作できるので、便利な機能です。

コロン演算子の使用法

MATLAB のコロン演算子 (:) の利用は、Java 配列参照のサブスクリプトでサポートされています。この演算子は、MATLAB 配列の内容を参照するときと同様に機能します。java.lang.Double オブジェクトの Java 配列を使うと、ステートメント

dblArray(2,2:4) は、低レベル配列 dblArray(2) の一部分を参照します。新規配列 row2Array は、列 2 から 4 の要素から作成されます。

```
dblArray
dblArray =
java.lang.Double[][]:
   [11]   [12]   [13]   [14]   [15]
   [21]   [22]   [23]   [24]   [25]
   [31]   [32]   [33]   [34]   [35]
   [41]   [42]   [43]   [44]   [45]

row2Array = dblArray(2,2:4)
row2Array =
java.lang.Double[]:
   [22]
   [23]
   [24]
```

「単一のサブスクリプトインデックスを使った配列のアクセス」で説明したように単一のサブスクリプトインデックスにおいてコロン演算子を使うことも可能です。サブスクリプトを数値ではなくコロンとすることにより、配列の配列を 1 つの線形配列に変換できます。つぎの例は、4 行 5 列の配列 dblArray を 20 行 1 列の線形配列に変換します。

```
linearArray = dblArray(:)
linearArray =
java.lang.Double[]:
   [11]
   [12]
   [13]
   [14]
   [15]
   [21]
   [22]
    .
    .
    .
```

これは、N 次元の Java 配列構造に対しても同様に機能します。コロン演算子を配列の単一のサブスクリプトインデックスとして使うと、オリジナルの配列のすべての要素で構成される線形配列を生成します。

注意　Java および MATLAB 配列は、メモリでは異なって格納されます。これは、線形配列内で与えられる順番に反映されます。Java 配列要素は、行列の行上記の配列の (11, 12, 13, ...) に一致する順番で格納されます。MATLAB 配列要素は、列 (11, 21, 31, ...) に一致する順番で格納されます。

サブスクリプトでの END の利用

アクセスステートメントの最初のサブスクリプトで end キーワードを使うことができます。最初のサブスクリプトは、マルチレベル Java 配列構造の最上位配列を参照します。

注意　低レベル配列で end を使うことは、これらの配列のでこぼこな性質のため、無効です (「Java 配列の形状」) を参照してください)。この場合は、一致する終端の値は得られません。

つぎの例は、Java 配列 dblArray の 3 行目から最後の行のデータを表示します。

```
last2rows = dblArray(3:end, :)
last2rows =
java.lang.Double[][]:
   [31]   [32]   [33]   [34]   [35]
   [41]   [42]   [43]   [44]   [45]
```

Java 配列の割り当て

MATLAB配列で行うのと本質的に同じ方法でJava配列のオブジェクトに値を割り当てます。Java および MATLAB 配列は、全く異なる構造ですが、同じコマンドシンタックスを使って割り当てたい要素を指定します、Java と MATLAB 配列の違いに関する情報は、「Java 配列の表現方法」を参照してください。

つぎの例は、行 3、列 2 の dblArray の要素に値 300 を割り当てます。Java では、これは dblArray[2][1] です。

```
dblArray(3,2) = java.lang.Double(300)
dblArray =
java.lang.Double[][]:
   [11]   [ 12]   [13]   [14]   [15]
```

```
[21]  [ 22]  [23]  [24]  [25]
[31]  [300]  [33]  [34]  [35]
[41]  [ 42]  [43]  [44]  [45]
```

同じシンタックスを使ってオブジェクトのデータフィールドの要素に割り当てます。「Java 配列の要素へのアクセス」の myMenuObj の例に続けて、つぎのように menuItemArray の 3 番目のメニューアイテムに割り当てます。

```
myMenuObj.menuItemArray(3) = java.lang.String('Save As...');
```

単一のサブスクリプトインデックスを使った配列の割り当て

単一の配列サブスクリプトを使って、2 次元以上の Java 構造体にインデックスを付けることができます。Java 配列で利用したこの機能の説明は、「単一のサブスクリプトインデックスを使った配列のアクセス」を参照してください。

単一のサブスクリプトインデックスを使って配列にも値を割り当てることが可能です。下記の例は、1 次元 Java 配列 onedimArray を 2 次元 Java 配列 dblArray の行に割り当てます。1 次元配列を作成することから始めます。

```
onedimArray = javaArray('java.lang.Double', 5);
for i = 1:5
   onedimArray(i) = java.lang.Double(100 * i);
   end
```

dblArray(3) は、dblArray の 3 行目に表示された 5 行 1 列配列を参照するので、同様の次元の 5 行 1 列の onedimArray を割り当てることができます。

```
dblArray(3) = onedimArray
dblArray =
java.lang.Double[][]:
   [ 11]  [ 12]  [ 13]  [ 14]  [ 15]
   [ 21]  [ 22]  [ 23]  [ 24]  [ 25]
   [100]  [200]  [300]  [400]  [500]
   [ 41]  [ 42]  [ 43]  [ 44]  [ 45]
```

線形配列の割り当て

配列構造を単一の線形配列のように扱うことによって、多次元 Java 配列の各要素に値を割り当てることができます。これは、MATLAB のコロン演算子を使った単一の数値サブスクリプトの置き換えを伴います。dblArray 配列から始める場合、つぎのステートメントを使って、2 次元配列の各オブジェクトの内容を初期化することができます。

```
dblArray(:) = java.lang.Double(0)
dblArray =
java.lang.Double[][]:
   [0]   [0]   [0]   [0]   [0]
   [0]   [0]   [0]   [0]   [0]
   [0]   [0]   [0]   [0]   [0]
   [0]   [0]   [0]   [0]   [0]
```

コロン演算子の利用

MATLAB配列の操作と同様にMATLABコロン演算子を使うことができます。下記のステートメントは、与えられた値を Java 配列 dblArray の 4 つの行の各々に割り当てます (各行は、実際には別々の Java 配列を現すことに注意してください)。

```
dblArray(1,:) = java.lang.Double(125);
dblArray(2,:) = java.lang.Double(250);
dblArray(3,:) = java.lang.Double(375);
dblArray(4,:) = java.lang.Double(500)
dblArray =
java.lang.Double[][]:
   [125]   [125]   [125]   [125]   [125]
   [250]   [250]   [250]   [250]   [250]
   [375]   [375]   [375]   [375]   [375]
   [500]   [500]   [500]   [500]   [500]
```

空行列の割り当て

MATLAB配列の操作時に、空行列([]で表わされる0行0列配列)を配列の要素に割り当てることができます。Java 配列に対しては、配列要素に [] を割り当てることも可能です。これは、0 行 0 列配列ではなく NULL 値を Java 配列要素に格納します。

サブスクリプトによる削除

MATLAB の行全体あるいは列全体に空行列を割り当てるとき、MATLAB は、実際は配列から影響される行あるいは列を削除します。下記の例では、空行列はMATLAB行列matlabArrayの4列目のすべての要素に割り当てられます。そのため、4 列目は、行列から完全に削除されます。これは、4 行 5 列から 4 行 4 列に次元を変更します。

```
matlabArray = [11 12 13 14 15; 21 22 23 24 25; ...
           31 32 33 34 35; 41 42 43 44 45]
matlabArray =
   11   12   13   14   15
   21   22   23   24   25
   31   32   33   34   35
   41   42   43   44   45


matlabArray(:,4) = []
matlabArray =
   11   12   13   15
   21   22   23   25
   31   32   33   35
   41   42   43   45
```

Java 配列に空行列を割り当てることは可能ですが、影響は異なります。つぎの例は、同じ演算を Java 配列に対して行ったときに、構造が壊れないことを示しています。4 行 5 列の次元を保持します。

```
dblArray(:,4) = []
dblArray =
java.lang.Double[][]:
   [125]   [125]   [125]     []   [125]
   [250]   [250]   [250]     []   [250]
```

```
[375]   [375]   [375]   []   [375]
[500]   [500]   [500]   []   [500]
```

dblArray データ構造体は、実際は java.lang.Double オブジェクトの 5 要素配列の配列です。空配列の割り当てで、低レベル配列の 4 番目の要素に NULL 値を配置します。

Java 配列の結合

Java 配列に対して行える結合の程度は、MATLAB と Java 配列との明確な構造の違いにより制限されます。MATLAB の多次元配列の Java 版は、1 次元 Java 配列の配列で構成されるかなり複雑な構造です。この構造の個々の低レベル配列は、長さが異なり、配列の構造全体が一様でなく、でこぼこである場合があります。Java 配列のこの性質は、配列間の不適合となる可能性があり、そのため MATLAB の結合演算によって結合される程度を制限します。

結合する Java 配列が同じクラスのとき (あるいは配列のクラスがもう一方の配列クラスに割り当て可能であるとき)、割り当てに互換性があると考えられます。この場合、結果の配列のクラスは構成要素の配列のすべてに互換性があるクラスです。そして、結果の配列の長さは、構成要素の配列の長さの総和です。

下記の java.awt.Frame の 2 つの配列を考えます。frmArray1 配列は、Frame オブジェクトの 3 行 5 列配列です。frmArray2 配列は、同じオブジェクトの 7 行 12 列配列です。cat コマンドを使って 2 つの配列を結合します。結果は、java.awt.Frame クラスの 10 要素配列です。

注意　cat コマンドの使用時に、第一引数 (結合を行う軸を指定します) は、1 でなければなりません。Java オブジェクトは、最初の軸に沿ってのみ結合されます。

```
frmArray1 = javaArray('java.awt.Frame', 3, 5);
frmArray2 = javaArray('java.awt.Frame', 7, 12);
twoFrames = cat(1, frmArray1, frmArray2);
```

```
class(twoFrames)
ans =
java.awt.Frame[][]

length(twoFrames)
ans =
10
```

構成要素の配列に割り当ての互換性がない場合、結果の配列はクラス java.lang.Object です。配列の長さは、結合によって連結された配列の数と等しくなります。これは、以下の例で示されます。

```
ptArray = javaArray('java.awt.Point', 8, 5);
dblArray = javaArray('java.lang.Double', 8, 5);

class([ptArray dblArray])
ans =
java.lang.Object[]

length([ptArray dblArray])
ans =
2
```

新規の配列参照の作成

MATLABのJava配列は*リファレンス*なので、配列変数を他の変数に割り当てると配列の 2 番目のリファレンスになります。

2 つの別個の配列変数が共通の配列を参照するつぎの例を考えます。オリジナルの配列 origArray が作成され、初期化されます。この配列変数のコピーは、ステートメント newArrayRef = origArray によって作成されます。newArrayRef によって参照される配列に対する変更は、オリジナル配列にも現れます。

```
origArray = javaArray('java.lang.Double', 3, 4);
for i = 1:3
   for j = 1:4
   origArray(i,j) = java.lang.Double((i * 10) + j);
   end
end
```

```
origArray
origArray =
java.lang.Double[][]:
   [11]   [12]   [13]   [14]
   [21]   [22]   [23]   [24]
   [31]   [32]   [33]   [34]

%  ----- Make a copy of the array reference -----
newArrayRef = origArray;
newArrayRef(3,:) = java.lang.Double(0);

origArray
origArray =
java.lang.Double[][]:
   [11]   [12]   [13]   [14]
   [21]   [22]   [23]   [24]
   [ 0]   [ 0]   [ 0]   [ 0]
```

Java 配列のコピーの作成

要素の部分を記述する配列 (あるいはサブ配列) にインデックスを付け、このサブ配列に変数を割り当てることによって、既存の Java 配列から全く新規の配列を作成することができます。割り当ては、オリジナル配列の値を新規配列の対応するセルにコピーします。

「新規の配列参照の作成」の例題のように、オリジナル配列が作成され、初期化されます。しかし、今回は、配列の参照をコピーするのではなく、配列の内容からコピーが作成されます。新規配列への参照を使って行われた変更は、オリジナルには影響を与えません。

```
origArray = javaArray('java.lang.Double', 3, 4);
for i = 1:3
   for j = 1:4
   origArray(i,j) = java.lang.Double((i * 10) + j);
   end
end

origArray
origArray =
java.lang.Double[][]:
   [11]   [12]   [13]   [14]
```

```
   [21]   [22]   [23]   [24]
   [31]   [32]   [33]   [34]

%  ----- Make a copy of the array contents -----
newArray = origArray(:,:);
newArray(3,:) = java.lang.Double(0);

origArray
origArray =
java.lang.Double[][]:
   [11]   [12]   [13]   [14]
   [21]   [22]   [23]   [24]
   [31]   [32]   [33]   [34]
```

Java メソッドにデータを渡す

MATLAB から Java コードへの呼び出しを行うとき、呼び出しで渡す MATLAB のデータタイプは、Java 言語ネイティブなデータタイプに変換されます。MATLAB は、既に Java オブジェクトである引数を除いて、渡されるすべての引数に対してこの変換を行います。本節では、MATLAB 固有のデータタイプに対して行われる変換について説明し、最後にオーバロードメソッドに対する呼び出しに引数が与える影響について説明します。

データが呼び出されるメソッドによって出力される場合、MATLAB は、このデータを受信し、必要ならば適切な MATLAB 書式に変換します。このプロセスは、「Java メソッドで出力されるデータの取り扱い」で説明します。

本節では、つぎのトピックスを説明します。

- MATLAB 引数データの変換
- 組み込みデータタイプを渡す
- 文字列引数を渡す
- Java オブジェクトを渡す
- その他のデータ変換のトピックス
- データをオーバロードメソッドに渡す

MATLAB 引数データの変換

Java メソッドに引数として渡される MATLAB データは、MATLAB によって、Java 言語データを最も適切に表わすデータタイプに変換されます。下記の表は、渡される引数に対するすべての MATLAB ベースのタイプと、入力引数に対して定義される Java ベースのタイプを示します。各行は、マッチに近い順番に左から右に、MATLAB タイプの後に一致する Java 引数を示します。MATLAB タイプ (セル配列以外) は、すべてのスカラ (1 行 1 列) 配列あるいは行列です。すべての Java タイプは、スカラ値あるいは配列です。

表 5-2: MATLAB タイプの Java タイプへの変換

MATLAB 引数	最も近いタイプ (7)	Java 入力引数 (スカラまたは配列)					最も近くないタイプ (1)
double (logical)	boolean	byte	short	int	long	float	double
double	double	float	long	int	short	byte	boolean

表 5-2: MATLAB タイプの Java タイプへの変換

MATLAB 引数	最も近いタイプ (7)	Java 入力引数 (スカラまたは配列)					最も近くないタイプ (1)
single	float	double					
char	String	char					
uint8	byte	short	int	long	float	double	
uint16	short	int	long	float	double		
uint32	int	long	float	double			
int8	byte	short	int	long	float	double	
int16	short	int	long	float	double		
int32	int	long	float	double			
cell array of strings	array of String						
Java object	Object						
cell array of object	array of Object						

Java コードに渡される引数のデータタイプ変換は、つぎの 3 つのカテゴリで説明します。MATLAB の各カテゴリの取り扱い方は異なります。

- 組み込みデータタイプを渡す
- 文字列引数を渡す
- Java オブジェクトを渡す

組み込みデータタイプを渡す

Javaは、言語固有でJavaオブジェクトとして表現されない8個のデータタイプをもちます。これらは、*組み込み*、あるいは基本的な データタイプとして参照され、boolean, byte, short, long, int, double, float, char を含みます。MATLAB は、独自のデータを表「MATLAB タイプから Java タイプへの変換」に従ってこれらの Java 組み込みタイプに変換します。組み込みタイプは、表の最初の 10 行にあります。

呼び出している Java メソッドがこれらのデータタイプのうちの 1 つを要求する場合、表の一番左の列の MATLAB 引数のタイプを渡すことができます。メソッドがこれらのタイプの配列を受け取る場合、そのデータタイプの MATLAB 配列を渡すことができます。MATLAB は、引数のデータタイプをメソッドの宣言で割り当てられたタイプに変換します。

下記の MATLAB コードは、最上位のウィンドウフレームを作成し、次元を設定します。setBounds の呼び出しは、4 つの double タイプの MATLAB スカラを、int タイプの 4 つの引数を受け取る継承された Java Frame メソッド setBounds に渡します。MATLAB は、呼び出しの前に各 64 ビット double データタイプを 32 ビット整数に変換します。以下は、setBounds メソッドの宣言と、メソッドを呼び出す MATLAB コードです。

```
public void setBounds(int x, int y, int width, int height)
```

```
frame=java.awt.Frame;
frame.setBounds(200,200,800,400);
frame.setVisible(1);
```

配列の組み込みタイプを渡す

組み込みタイプの配列 として定義された引数を使って Java メソッドを呼び出すと、互換性のあるベースタイプをもつ MATLAB 行列を作成して渡すことができます。つぎのコードは、4 つの x と y 座標を Polygon コンストラクタに渡して多角形を定義します。2 つの double の 1 行 4 列 MATLAB 配列は、最初の 2 つの引数が整数配列であると考えられる java.awt.Polygon に渡されます。以下は、Java メソッドの宣言と、メソッドを呼び出し、座標を検証する MATLAB コードです。

```
public Polygon(int xpoints[], int ypoints[], int npoints)
```

```
poly = java.awt.Polygon([14 42 98 124], [55 12 -2 62], 4);
```

```
[poly.xpoints poly.ypoints]      % Verify the coordinates
ans =
14    55
42    12
98    -2
124    62
```

MATLAB 配列は値によって渡されます

MATLAB 配列は値によって渡されるので、それらに対して Java メソッドが行う変更は、MATLAB コードでは見ることができません。Java メソッドが配列に対して行う変更にアクセスする必要がある場合は、MATLAB 配列を渡すよりも、リファレンスである Java 配列を作成して渡すべきです。MATLAB での Java 配列の利用に関する説明は、「Java 配列の利用」を参照してください。

注意　一般に、引数リスト内のデータのリファレンスを渡すのとは対照的に、メソッドが出力引数のメカニズムを使って変更されたデータを出力することが望ましいです。

文字列引数を渡す

クラス java.lang.String のオブジェクトとして定義された引数を持つ Java メソッドを呼び出すには、以前の Java 呼び出しから出力された String オブジェクトや、MATLABの1行n列キャラクタ配列を渡すことができます。キャラクタ配列を渡す場合は、MATLAB は配列を java.lang.String の Java オブジェクトに変換します。

プログラミングの例題として、「例題 - URL の読み込み」を参照してください。これは、Java URL クラスコンストラクタに渡される URL を持つ MATLAB キャラクタ配列を示しています。下記のコンストラクタは、Java String引数を要求します。

```
public URL(String spec) throws MalformedURLException
```

このコンストラクタの MATLAB の呼び出しにおいて、URL を指定するキャラクタ配列が渡されます。MATLAB は、コンストラクタの呼び出しの前に。この配列を Java String オブジェクトに変換します。

```
url = java.net.URL(...
    'http://www.ncsa.uiuc.edu/demoweb/url-primer.html')
```

配列の文字列を渡す

呼び出しているメソッドがタイプ String の配列の引数を要求する場合、文字列を MATLAB セル配列にパッケージ化することによって、そのような配列を作成することができます。別々の配列のセルにそれらを格納しているため、文字列は長さが異なる場合があります。メソッド呼び出しの一部分として、MATLAB はセル配列を String オブジェクトの Java 配列に変換します。

つぎの例で、ユーザ定義クラスの echoPrompts メソッドは、MATLAB がオリジナルの書式から変換した文字列配列の引数を、文字列のセル配列として持ちます。Java メソッドのパラメータリストは、つぎのように表わされます。

```
public String[] echoPrompts(String s[])
```

両方の文字列を MATLAB セル配列に格納することによって、入力引数を作成します。MATLAB は、この構造体を String の Java 配列に変換します。

```
myaccount.echoPrompts({'Username: ','Password: '})
ans =
'Username: '
'Password: '
```

Java オブジェクトを渡す

特定の Java クラスに属する引数を持つメソッドを呼び出すとき、そのクラスのインスタンスのオブジェクトを渡す必要があります。下記の例で、java.awt.Menu クラスに属する add メソッドは、引数として java.awt.MenuItem クラスのオブジェクトが必要です。このメソッド宣言は、以下の通りです。

```
public MenuItem add(MenuItem mi)
```

この例は、「組み込みデータタイプを渡す」の説明の例で作成されたフレームに対して操作します。このコードの 2,3,4 行目は、既存のウィンドウフレームに貼り付けるメニューのアイテムを追加します。menu1.add の呼び出しの各々において、java.awt.MenuItem Java クラスであるオブジェクトが渡されます。

```
menu1 = java.awt.Menu('File Options');
```

```
menu1.add(java.awt.MenuItem('New'));
menu1.add(java.awt.MenuItem('Open'));
menu1.add(java.awt.MenuItem('Save'));

menuBar=java.awt.MenuBar;
menuBar.add(menu1);
frame.setMenuBar(menuBar);
```

クラス java.lang.Object のオブジェクトの取り扱い

呼び出されるメソッドが java.lang.Object クラスの引数を持つときには、特殊な場合が存在します。このクラスは、Java クラス階層のルートなので、引数内の任意のクラスのオブジェクトを渡すことができます。以下のハッシュテーブルの例は、異なるクラスに属するオブジェクトを、java.lang.Object の引数を要求する共通のメソッド put に渡します。put に対するメソッドの宣言は、以下の通りです。

```
public synchronized Object put(Object key, Object value)
```

つぎの MATLAB コードは、異なるタイプのオブジェクト（boolean, float, string）を put メソッドに渡します。

```
hTable = java.util.Hashtable;
hTable.put(0, java.lang.Boolean('TRUE'));
hTable.put(1, java.lang.Float(41.287));
hTable.put(2, java.lang.String('test string'));

hTable              % ハッシュテーブルの内容確認
hTable =
{1.0=41.287, 2.0=test string, 0.0=true}
```

java.lang.Object を受け取るメソッドに引数を渡すときには、組み込みデータタイプのオブジェクトに対するクラス名を指定する必要はありません。上記の例の 3 行目は、41.287 はクラス java.lang.Float であることを指定します。これは省略が可能で、下記の例のように単に 41.287 とすることができます。そのため、MATLAB は各引数に対して最もマッチする Java オブジェクト表現を選択して、各オブジェクトを作成します。

上記の例の 3 回の put の呼び出しは、つぎのように書き直すことができます。

```
hTable.put(0, 1);
hTable.put(1, 41.287);
hTable.put(2, 'test string');
```

配列にオブジェクトを渡す

Java メソッドに渡すことが可能な Java オブジェクト配列には、Java 配列と MATLAB セル配列の 2 つのタイプがあります。

オブジェクトが既に配列内に置かれている場合は (Java コンストラクタから出力される配列あるいは javaArray 関数によって MATLAB で作成された配列)、呼び出されるメソッドの引数のように、単純に渡します。これは、既に Java 配列なので、MATLAB は変換を行いません。

Java 配列にないオブジェクトを持つ場合は、MATLAB は、MATLAB セル配列にそれらを渡すことができます。この場合、MATLAB は引数を渡す前にセル配列を Java 配列に変換します。

つぎの例は、クラス java.awt.Point の配列を受け取るユーザ定義クラスの mapPoints メソッドを示します。これに対するメソッドの宣言は、以下の通りです。

```
public Object mapPoints(java.awt.Point p[])
```

下記の MATLAB コードは、4 つの Java Point オブジェクトを含む 2 行 2 列のセル配列を作成します。セル配列が mapPoints メソッドに渡されるとき、MATLAB はセル配列をタイプ java.awt.Point の Java 配列に変換します。

```
pointObj1 = java.awt.Point(25,143);
pointObj2 = java.awt.Point(31,147);
pointObj3 = java.awt.Point(49,151);
pointObj4 = java.awt.Point(52,176);

cellArray={pointObj1, pointObj2; pointObj3, pointObj4}
cellArray =
   [1x1 java.awt.Point]    [1x1 java.awt.Point]
   [1x1 java.awt.Point]    [1x1 java.awt.Point]

testData.mapPoints(cellArray);
```

Java オブジェクトのセル配列の取り扱い

MATLAB シンタックス {a1,a2,...} を使って Java オブジェクトのセル配列を作成します。通常の方法であるシンタックス a{m,n,...} で、Java オブジェクトのセル配列にインデックスを付けます。

つぎの例は、2 つの Frame オブジェクト、frame1 と frame2 のセル配列を作成し、変数 frames を割り当てます。

```
frame1 = java.awt.Frame('Frame A');
frame2 = java.awt.Frame('Frame B');

frameArray = {frame1, frame2}
frameArray =
[1x1 java.awt.Frame]    [1x1 java.awt.Frame]
```

つぎのステートメントは、セル配列 frameArray の要素 {1,2} を変数 f に割り当てます。

```
f = frameArray {1,2}
f =
java.awt.Frame[frame2,0,0,0x0,invalid,hidden,layout =
java.awt.BorderLayout,resizable,title=Frame B]
```

その他のデータ変換のトピックス

MATLAB がデータを Java タイプに変換する方法に関して、その他にも興味深い項目がいくつか残っています。これは、MATLAB がどのように配列の次元を一致させるかや、空行列や空文字列をどのように扱うかを含みます。

配列の次元の変換への影響

本節で用いる用語 *dimension* は、長さ、幅、高さの特性ではなく、配列の要素を表わすため必要なサブスクリプトの数を表わします。たとえば、5 行 1 列配列は、個々の要素が 1 つの配列のサブスクリプトのみを使ってインデックス付けられる 1 次元として参照されます。

MATLAB の Java 配列への変換において、MATLAB は次元を特別な方法で扱います。MATLAB 配列に対して、次元は、配列内の非シングルトンの数として考えられます。たとえば、10 行 1 列配列は次元 1 で、1 行 1 列配列は次元 0 です。Java では、次元は単に入れ子形式の配列の数によって決定されます。たとえば、double[][] は次元 2 で、double は次元 0 です。

Java配列の次元数がMATLAB配列の次元数nと正確に一致する場合、変換の結果はn 次元の Java 配列になります。Java 配列が n よりも小さい次元である場合、残りの次元数が Java 配列内の次元数と一致するまで、変換は最初に 1 から始まるシングルトンの次元を落とします。

空行列と Null

空行列は、NULL が Java において有効な値である任意のメソッド引数に対して適用可能です。MATLAB の空文字列 ('') は、Java では空 (NULL でない)String オブジェクトに変換されます。

データをオーバロードメソッドに渡す

Java オブジェクトに対してオーバロードメソッドを呼び出すとき、MATLAB は、ユーザ呼び出しが渡す引数とメソッドに対して定義された引数を比較することによって、呼び出すメソッドを決定します。この説明では、メソッドはコンストラクタを含みます。呼び出すメソッドを決定するとき、MATLAB はオブジェクト配列またはセル配列に関する変換以外について、Java の変換規則に従って呼び出し引数を Java メソッドタイプに変換します。「配列にオブジェクトを渡す」を参照してください。

呼び出すメソッドを決定する方法

ユーザの MATLAB 関数が、Java メソッドを呼び出す場合、MATLAB は、

1. オブジェクト (またはスタティックメソッドに対してはクラス) がその名前のメソッドをもつことを確認します。
2. 呼び出しが、少なくとも 1 つのその名前のメソッドの引数と同数の引数を渡すかどうかを決定します。
3. 渡される各引数が、メソッドに対して定義された Java タイプに変換されることを確認します。

上記の全ての条件が満たされる場合は、MATLAB はメソッドを呼び出します。

オーバーロードされたメソッドの呼び出しにおいて、2 つ以上の候補がある場合は、MATLAB は呼び出し引数にもっとも合う引数を持つものを選択します。最初に、MATLAB は、渡される引数と互換性のない引数をもつメソッドを除外します (たとえば、メソッドが double の引数を持ち、渡される引数が char である場合)。

残りのメソッドの中で、MATLAB は、すべての引数の適合度の値の総和である、最高の適合度を持つものを選択します。各引数に対する適合度の値は、ベースタイプの適合度から MATLAB 配列の次元と Java 配列の次元との差分を引いたものです (配列の次元は、「配列の次元の変換への影響」で説明します)。2 つのメソッドが同じ適合度値をもつ場合は、Java クラスで最初に定義されたメソッドが選択されます。

例題 - オーバーロードされたメソッドの呼び出し

関数が java.io.OutputStreamWriter オブジェクト osw を作成し、オブジェクトについてメソッドを呼び出すと仮定します。

```
osw.write('Test data', 0, 9);
```

MATLAB は、クラス java.io.OutputStreamWriter が、3 つの write メソッドを定義することを見つけます。

```
public void write(int c);
public void write(char[] cbuf, int off, int len);
public void write(String str, int off, int len);
```

最初の write メソッドは引数を 1 つだけを受け取るので、MATLAB はそれを拒否します。その後、MATLAB は残りの 2 つの write メソッドの適合度を評価します。これらは、以下で説明する第一引数のみが異なります。

これらの 2 つのうちの最初の write メソッドにおいて、第一引数は、ベースタイプ char によって定義されます。表「MATLAB タイプから Java タイプへの変換」は、呼び出し引数のタイプ (MATLAB の char) に対して、Java タイプ char は値 6 を持つことを示します。呼び出し引数と Java 引数の次元の違いはありません。そのため、第一引数の適合度値は 6 です。

他の write メソッドにおいて、第一引数は Java タイプの String で、適合度値は 7 です。Java 引数の次元は 0 なので、Java 引数と呼び出し引数との次元の差は 1 です。そのため、第一引数に対する適合度値は 6 です。

これらの 2 つの write メソッドに対する適合度値は等しいので、MATLAB は (char[] 第一引数を持つ) クラス定義で最初にリストされたメソッドを呼び出します。

Java メソッドから出力されるデータの扱い

多くの場合で、Java から出力されるデータは、MATLAB 内部で操作されたデータタイプとの互換性がありません。この場合、MATLAB は出力値を MATLAB 言語ネイティブなデータタイプに変換します。本節では、Java メソッドの呼び出しから出力される様々なデータタイプにおいて実行される変換について説明します。

本節では、以下のトピックスを説明します。

- Java 出力値の変換
- 組み込みデータタイプ
- Java オブジェクト
- オブジェクトから MATLAB データタイプへの変換

Java 出力値の変換

つぎの表は、Java 出力タイプと結果の MATLAB タイプの一覧です。Java ベースの出力タイプによっては、つぎの表に示すように、スカラと配列の出力値の取り扱いが異なることがあります。

表 5-3: Java タイプから MATLAB タイプへの変換

Java 戻り値	スカラを出力する場合の結果の MATLAB タイプ	配列を出力する場合の結果の MATLAB タイプ
boolean	double (logical)	double (logical)
byte	double	int8
short	double	int16
int	double	int32
long	double	double
float	double	single
double	double	double
char	char	char

組み込みデータタイプ

Java *組み込み*データタイプは、「組み込みデータタイプを渡す」で説明します。基本的に、このデータタイプは、 boolean, byte, short, long, int, double, float, char を含みます。メソッド呼び出しから出力される値がこれらの Java 組み込みタイプのいずれかであるとき、MATLAB は表「Java タイプから MATLAB タイプへの変換」に従って変換します。

単一の数値あるいは boolean 値は、double の 1 行 1 列行列に変換します。これは、MATLAB での利用に対して便利です。数値あるいは boolean 出力値の配列は、必要な容量を最小化するため、最も近いベースタイプの配列に変換します。配列の変換は、表の右列に示されています。

Java タイプ char の出力値は、char の 1 行 1 列行列に変換されます。Java char の配列は、そのタイプの MATLAB 配列に変換されます。

Java オブジェクト

メソッド呼び出しが Java オブジェクトを出力するとき、MATLAB は、それらの形式をオリジナルのままにします。Java オブジェクトとして残るので、他の Java メソッドと相互に作用させるためにそれらを継続利用することができます。

唯一の例外は、メソッドがタイプ java.lang.Object のデータを出力するときです。このクラスは、Java クラス階層のルートで、様々なタイプのオブジェクトや配列に対して catchall として頻繁に利用されます。呼び出されるメソッドが Object クラスの値を出力するとき、MATLAB は、その値を表「Java タイプから MATLAB タイプへの変換」に従って変換します。つまり、java.lang.Integer、あるいは、java.lang.Boolean のような数値および boolean オブジェクトは、double の 1 行 1 列 MATLAB 行列に変換されます。これらのタイプのオブジェクト配列は、表の右側の列の MATLAB データタイプに変換されます。その他のオブジェクトタイプは、変換されません。

オブジェクトを MATLAB データタイプに変換

クラス String およびクラス Object のオブジェクトの例外においては、MATLAB は、ネイティブな MATLAB データタイプのメソッド呼び出しから出力された Java オブジェクトを変換しません。Java オブジェクトデータを MATLAB で簡単で使いやすい形式に変換したい場合は、そのために利用可能な MATLAB 関数が存在します。これらは、つぎの節で説明します。

MATLAB double データタイプへの変換

MATLABで double コマンドを使って、任意の Java オブジェクトあるいはオブジェクトの配列を MATLAB の double データタイプに変換することができます。double コマンドにより行われるアクションは、指定したオブジェクトのクラスにより異なります。

- オブジェクトが数値クラス (java.lang.Number あるいはそのクラスから継承するクラスのいずれか) のインスタンスである場合は、MATLAB はプリセット変換アルゴリズムを使ってオブジェクトを MATLAB の double に変換します。
- オブジェクトが数値クラスでない場合は、MATLAB は toDouble というメソッドを実現するかどうかについてクラス定義をチェックします。MATLAB は、toDoubleを使ってJavaオブジェクトからMATLABのdoubleデータタイプへの変換を行います。そのようなメソッドが、このクラスに対して実現される場合は、MATLAB は実行して変換を行います。
- ユーザが独自に作成したクラスを利用する場合は、独自の toDouble メソッドを作成して、そのクラスのオブジェクトから MATLAB double への変換を行うことができます。これにより、ユーザのクラスに属するオブジェクトに対するデータタイプの変換の手段を指定することができます。

注意　指定したオブジェクトのクラスが、java.lang.Number でなく、java.lang.Number から継承されず、toDouble ソッドを実現しない場合は、double コマンドを使ってオブジェクトを変換しようとすると、MATLAB でエラーになります。

---

double コマンドの構文は、以下に示す通りです。ここで、object は Java オブジェクトまたは Java オブジェクトの配列です。

```
double(object);
```

MATLAB char データタイプへの変換

MATLABのcharコマンドを使って、java.lang.Stringオブジェクトと配列をMATLABデータタイプに変換することができます。単一の java.lang.String オブジェクトは、MATLAB キャラクタ配列に変換されます。java.lang.String オブジェクトの配列は、各セルがキャラクタ配列である MATLAB セル配列に変換されます。

char コマンドで指定されたオブジェクトが、java.lang.String クラスでない場合は、MATLAB は、toChar というメソッドを実現するかどうかについてそのクラスをチェックします。実現する場合は、MATLAB はクラス toChar のメソッドを実行

して変換を行います。ユーザ独自の定義を作成する場合は、ユーザの必要に応じて変換を実行する toChar メソッドを作成することにより、この機能を利用することができます。

注意　指定したオブジェクトのクラスが、java.lang.String でなく、toChar メソッドを実現しない場合は、char コマンドを使ってオブジェクトを変換しようとすると、MATLAB でエラーになります。

---

char コマンドの構文は、以下に示す通りです。ここで、object は Java オブジェクトあるいは Java オブジェクトの配列です。

```
char(object);
```

MATLAB 構造体への変換

Java オブジェクトは、オブジェクトの特性の多くがオブジェクト内で定義されたフィールド名によってアクセス可能である点が MATLAB 構造体と同じです。MATLABでのデータの扱いを容易にするため、Javaオブジェクトを MATLAB構造体に変換したい場合があります。そのためには、オブジェクトに対して MATLAB の struct コマンドを使ってください。

struct コマンドの構文は、以下に示す通りです。ここで、object は Java オブジェクトあるいは、Java オブジェクトの配列です。

```
struct(object);
```

つぎの例は、java.awt.Polygon オブジェクトを MATLAB 構造体に変換します。MATLAB の構造体の演算を使ってオブジェクトのフィールドに直接アクセスできます。最終行は、三番目の配列要素に新しい値を置くために、配列 pstruct.xpoints にインデックスを付けます。

```
polygon = java.awt.Polygon([14 42 98 124], [55 12 -2 62], 4);

pstruct = struct(polygon)
pstruct =
   npoints: 4
   xpoints: [4x1 int32]
   ypoints: [4x1 int32]
```

```
pstruct.xpoints
ans =
   14
   42
   98
  124

pstruct.xpoints(3) = 101;
```

MATLAB セル配列への変換

cellコマンドを使ってJava配列あるいはJavaオブジェクトをMATLABセル配列に変換します。結果のセル配列の要素は、Java 配列要素または Java オブジェクトに最も近い MATLAB タイプ (存在すれば) になります。

cell コマンドの構文は、以下の通りです。ここで、object は Java オブジェクトまたはオブジェクトの Java 配列です。

```
cell(object);
```

下記の例で、MATLAB セル配列は、各セルが異なるデータタイプの配列を保持して作成されます。1 行目で使われる cell コマンドは、各タイプのオブジェクト配列をセル配列に変換します。

```
import java.lang.* java.awt.*;

%  double の Java 配列を作成
dblArray = javaArray('java.lang.Double', 1, 10);
for m = 1:10
  dblArray(1, m) = Double(m * 7);
end

%  points の Java 配列を作成
ptArray = javaArray('java.awt.Point', 3);
ptArray(1) = Point(7.1, 22);
ptArray(2) = Point(5.2, 35);
ptArray(3) = Point(3.1, 49);

%strings の Java 配列を作成
strArray = javaArray('java.lang.String', 2, 2);
strArray(1,1) = String('one');    strArray(1,2) = String('two');
strArray(2,1) = String('three');  strArray(2,2) = String('four');
```

```
% セル配列に変換
cellArray = {cell(dblArray), cell(ptArray), cell(strArray)}
cellArray =
   {1x10 cell}   {3x1 cell}   {2x2 cell}

cellArray{1,1}      % double タイプ配列
ans =

  [7]  [14]  [21]  [28]  [35]  [42]  [49]  [56]  [63]  [70]

cellArray{1,2}      % Java.awt.Point タイプ配列

ans =

   [1x1 java.awt.Point]
   [1x1 java.awt.Point]
   [1x1 java.awt.Point]

cellArray{1,3}      % char array タイプ配列

ans =

   'one'    'two'
   'three'  'four'
```

プログラミングの例題

Javaクラスおよびオブジェクトとの MATLAB インタフェースのデモを行う 4 種類のプログラミングの例題を示します。例題は、以下の通りです。

- 例題 - URL の読み込み
- 例題 - IP アドレスの検索
- 例題 - シリアルポートによる通信
- 例題 - Phone Book の作成と使用

例題は、つぎの一般的なフォーマットで表わされています。

- 概要 - 例題が行うこととそのための Java インタフェースの使用の概要です。強調されているのは、例題コードで作成され使用される重要な Java オブジェクトです。
- 説明 - 例題内のコードの詳しい説明です。大きい関数については、説明は数個のステートメントに注目して機能別に分けられています。
- 例題の実行 - 例題コードの実行の出力例です。

例題の説明は、Java 関連の関数に集中しています。例題で使われている他の MATLAB プログラミングの構成、演算子、関数については、MATLAB ドキュメントの適切な部分を参照してください。

例題 - URL の読み込み

このプログラム URLdemo は、そのサイトでファイルからテキストを読み込むために、URL (Uniform Resource Locator) で指定された web サイトへの接続をオープンします。便利な URL の操作を可能にする Java API クラス、java.net.URL のオブジェクトを作成します。その後、URL オブジェクトについてメソッドを呼び出し、接続をオープンします。

サイトからテキスト行を読み込み表示するために、URLdemo は Java I/O パッケージ java.io からクラスを利用します。InputStreamReader オブジェクトを作成し、そのオブジェクトを使って BufferedReader オブジェクトを作成します。最後に、BufferedReader オブジェクトについてメソッドを呼び出し、指定した行数をサイトから読み込みます。

URLdemo の説明

URLdemo で行われる主な作業は、以下の通りです。

1．URL オブジェクトの作成

例題は、最初に java.net.URL についてコンストラクタを呼び出し、結果のオブジェクトを変数 url に割り当てます。URL コンストラクタは、アクセスされる URL 名を文字列引数として 1 つもちます。コンストラクタは、入力 URL が有効な形式かどうかをチェックします。

```
url = java.net.URL(...
    'http://www.ncsa.uiuc.edu/demoweb/url-primer.html')
```

2．URL への接続のオープン

例題の 2 番目のステートメントは、URL オブジェクト url についてメソッド openStreamを呼び出し、オブジェクトで指定されたwebサイトとの接続を確立します。メソッドは、サイトからのバイトの読み込み用に InputStream オブジェクトを変数 is に出力します。

```
is = openStream(url)
```

3．バッファされたストリームリーダの設定

つぎの 2 行は、キャラクタに対してバッファされたストリームリーダを作成します。java.io.InputStreamReader コンストラクタは、入力ストリーム is によって呼び出され、キャラクタの読み込みが可能なオブジェクトである変数 isr に出力します。その後、jjava.io.BufferedReader コンストラクタは、isr により呼び出され、

BufferedReader オブジェクトを変数 br に出力します。バッファされたリーダは、キャラクタ、配列、行の効果的な読み込みのために提供されています。

```
isr = java.io.InputStreamReader(is)
br = java.io.BufferedReader(isr)
```

4．テキスト行の読み込みと表示

最後のステートメントは、サイトからテキストの最初の 10 行を読み込み、表示します。10 回繰り返す MATLAB の for ステートメント内部では、BufferedReader メソッド readLine はサイトから (return または改行キャラクタによって終了する) テキスト行を読み込みます。MATLAB で出力を表示するには、ステートメントの終端にセミコロンを使わずに変数 s に割り当てます。

```
for i = 1:10
      s = readLine(br)
end
```

例題の実行

例題の実行時に、以下のような出力が表示されます。

```
url =
http://www.ncsa.uiuc.edu/demoweb/url-primer.html
is =
java.io.BufferedInputStream@62d
isr =
java.io.InputStreamReader@67d
br =
java.io.BufferedReader@644
s =
<TITLE>A Beginner’s Guide to URLs</TITLE>
s =
<H1>A Beginner’s Guide to URLs</H1>
s =
    ”
s =
What is a URL?  A URL is a <b>Uniform Resource Locator</b>.  Think
s =
of it as a networked extension of the standard <i>filename</i>
```

```
s =
concept: not only can you point to a file in a directory, but that s =
file and that directory can exist on any machine on the network,
s =
can be served via any of several different methods, and might not s =
even be something as simple as a file: URLs can also point to
s =
queries, documents stored deep within databases, the results of a s =
<i>finger</i> or <i>archie</i>
```

例題 - IP アドレスの検索

関数 resolveip は、IP(internet protocol) ホスト名またはアドレスのいずれかを出力します。resolveip にホスト名を渡す場合は、IP アドレスを出力します。resolveip に IP アドレスを渡す場合は、ホスト名を出力します。関数は、Java API クラス java.net.InetAddress を利用します。これを使って、DNS コールを行わずにホスト名に対する IP アドレスを求めたり、あるいは与えられた IP アドレスに対するホスト名を求めることができます。

resolveip は InetAddress クラスについて static を呼び出して InetAddress オブジェクトを取得します。その後、InetAddress オブジェクトについて acccessor メソッドを呼び出して入力引数に対するホスト名または IP アドレスを取得します。プログラムの入力引数に応じて、ホスト名あるいは IP アドレスを表示します。

resolveip の説明

resolveip が行う主な作業は、以下の通りです。

1．InetAddress オブジェクトの作成

java.net.InetAddress クラスは、コンストラクタの代わりに、クラスのインスタンスを出力する static メソッドを持ちます。try ステートメントは、ユーザが resolveip に渡した入力引数を渡して、メソッドのうちの 1 つの getByName を呼び出します。入力引数は、ホスト名または IP アドレスです。getByName が失敗すると、catch ステートメントは error メッセージを表示します。

```
function resolveip(input)
try
    address = java.net.InetAddress.getByName(input);
catch
    error(sprintf('Unknown host %s.', input));
end
```

2．ホスト名および IP アドレスの取得

例題では、java.net.InetAddress オブジェクトについて getHostName および getHostAddress アクセサ関数を呼び出して、それぞれホスト名と IP アドレスを取得します。

```
hostname = address.getHostName;
ipaddress = address.getHostAddress;
```

3．ホスト名または IP アドレスの表示

例題は、MATLAB の strcmp 関数を使って入力引数と IP アドレスを比較します。一致する場合は、MATLAB はインターネットアドレスに対するホスト名を表示します。入力が一致しない場合は、MATLAB は IP アドレスを表示します。

```
if strcmp(input,ipaddress)
    disp(sprintf('Host name of %s is %s', input, hostname));
else
    disp(sprintf('IP address of %s is %s', input, ipaddress));
end;
```

例題の実行

ホスト名を使って、関数 resolveip を呼び出す例を以下に示します。

```
resolveip www.mathworks.com
IP address of www.mathworks.com is 144.212.100.10
```

IP アドレスを使って、関数を呼び出します。

```
resolveip 144.212.100.10
Host name of 144.212.100.10 is www.mathworks.com
```

例題 - シリアルポートによる通信

プログラム serialexample は、通信ポートへのアクセスをサポートする Java API javax.comm パッケージのクラスを使います。ポート設定変数を定義した後で、serialexample は javax.comm.CommPortIdentifier オブジェクトを作成して、シリアル通信ポートを管理します。プログラムは、オブジェクトについて open メソッドを呼び出して、Tektronix oscilloscope(例題はオシロスコープを使わずに実行できます) に接続すると仮定される COM1 シリアルポートの低レベルインタフェースを記述する javax.comm.SerialPort ラスのオブジェクトを出力します。プログラム serialexample は、その後 SerialPort オブジェクトについていくつかのメソッドを呼び出してシリアルポートを設定します。

プログラム serialexample は、I/O パッケージ java.io を使ってシリアルポートとのやりとりを行います。static メソッドを呼び出してシリアルポートに対する OutputStream オブジェクトを出力します。その後、そのオブジェクトを java.io.OutputStreamWriter に対するコンストラクタに渡します。OutputStreamWriter オブジェクトについて write メソッドを呼び出して、オシロスコープにコントラストを設定するシリアルポートにコマンドを作成します。再度 write を呼び出してコントラストをチェックするコマンドを作成します。その後、java.io.InputStreamWriter クラスのオブジェクトを作成してシリアルポートから読み込みます。

SerialPort オブジェクトについて別の static メソッドを呼び出して、シリアルポートに対する OutputStream オブジェクトを出力します。そのオブジェクトについてメソッドを呼び出してポートから読み込みバイト数を取得します。InputStream オブジェクトを java.io.OutputStreamWriter に対するコンストラクタに渡します。その後、読み込むデータがある場合は、OutputStreamWriter オブジェクトについて read メソッドを呼び出して、オシロスコープが出力するコントラストデータを読み込みます。

注意　MATLAB は、シリアルポート I/O で説明した組み込みシリアルポートのサポートも提供しています。

例題の説明

serialexample が行う主な作業は、以下の通りです。

1．シリアルポートの設定と出力に対する変数の定義

最初の 5 個のステートメントは、シリアルポートの設定用の変数を定義します。最初のステートメントはボーレートを 9600 に定義し、2 番目はデータのビット数を 8 に定義し、3 番目は終了ビット数を 1 に定義します。4 番目のステートメントはパリティをオフに定義し、5 番目のステートメントはフローコントロール (handshaking) をオフに定義します。

```
SerialPort_BAUD_9600 = 9600;
SerialPort_DATABITS_8 = 8;
SerialPort_STOPBITS_1 = 1;
SerialPort_PARITY_NONE = 0;
SerialPort_FLOWCTRL_NONE = 0;
```

最後の変数定義は、シリアルポートへの書き込み用の終了キャラクタをキャリッジリターンに設定します。

```
terminator = char(13);
```

2．CommPortIdentifier オブジェクトの作成

javax.comm.CommPortIdentifier クラスは、コンストラクタの代わりに、クラスのインスタンスを出力する static メソッドを持ちます。例題は、これらのうちの 1 つ getPortIdentifierを呼び出して、ポートCOM1に対してCommPortIdentifierオブジェクトを出力します。

```
commPort = ... javax.comm.CommPortIdentifier.getPortIdentifier('COM1');
```

3．シリアルポートのオープン

例題は、CommPortIdentifier オブジェクト commPort について open を呼び出すことにより、シリアルポートをオープンします。open の呼び出しは、serialPort に割り当てる serialPort オブジェクトを出力します。open の第一引数はポート名 (オーナ) で、第二引数はポート名、第三引数はオープンの待ち時間 (ミリ秒) です。

```
serialPort = open(commPort, 'serial', 1000);
```

4．シリアルポートの設定

つぎの 3 つのステートメントは、SerialPort オブジェクト SerialPort についての設定メソッドを呼び出します。最初のステートメントは、setSerialPortParams を呼び出してボーレート、データビット、終了ビット、パリティを設定します。つぎの 2 つのステートメントは、setFlowControlMode を呼び出してフローコントロールを設定し、その後 enableReceiveTimeout を呼び出して受信データに対するタイムアウトを設定します。

```
setSerialPortParams(serialPort, SerialPort_BAUD_9600,...
SerialPort_DATABITS_8, SerialPort_STOPBITS_1,...
SerialPort_PARITY_NONE);
setFlowControlMode(serialPort, SerialPort_FLOWCTRL_NONE);
enableReceiveTimeout(serialPort, 1000);
```

5．出力ストリームライタの設定

例題は、つぎにコンストラクタを呼び出して、java.io.OutputStreamWriter を作成してオープンします。コンストラクタの呼び出しは、getOutputStream メソッド serialPort の呼び出しによって出力される java.io.OutputStream オブジェクトを渡して、OutputStreamWriter オブジェクトを out に割り当てます。

```
out = java.io.OutputStreamWriter(getOutputStream(serialPort));
```

6．シリアルポートにデータを書き出し出力ストリームをクローズ

例題は、オブジェクト out について write を呼び出すことにより、シリアルポートに文字列を書き出します。文字列は、機器が要求するコマンドターミネータを使ってオシロスコープのコントラストを 45 に設定するコマンドを (MATLAB の [ ] により) 結合することにより作成されます。つぎのステートメントは、out について flush を呼び出して、出力ストリームをフラッシュします。

```
write(out, ['Display:Contrast 45' terminator]);
flush(out);
```

その後、例題は再度 out について write を呼び出して、他の文字列をシリアルポートに送信します。この文字列は、コマンドターミネータによって連結されたオシロスコープのコントラスト設定を決定するための query コマンドです。その後例題は、出力ストリームについて close を呼び出します。

```
write(out, ['Display:Contrast?' terminator]);
close(out);
```

7．入力ストリームをオープンし、読み込むバイト数を決定

コントラストのクエリーに応じてオシロスコープから予想されるデータを読み込むために、例題は、static メソッド InputStream.getInputStream を呼び出してシリアルポートに対する InputStream オブジェクトを取得することによって、入力ストリームをオープンします。その後、例題は、InputStream オブジェクト in についてメソッド available を呼び出し、出力されたバイト数を numAvail に設定します。

```
in = getInputStream(serialPort);
numAvail = available(in);
```

8．シリアルポートに対する入力ストリームリーダの作成

例題は、その後 InputStream オブジェクト in を使ってコンストラクタ java.io.InputStreamReader を呼び出し、新規オブジェクトを reader に割り当てます。

```
reader = java.io.InputStreamReader(in);
```

9．シリアルポートからデータを読み込みリーダをクローズ

例題は、各バイトに対する InputStreamReader オブジェクトリーダについて read メソッドを呼び出すことによって、シリアルポートから読み込みます。read ステートメントは、MATLAB の配列結合を利用して、新規に読み込んだバイトを既に読み込んだバイトの配列に付加します。データを読み込んだ後で、例題は reader について close を呼び出して入力ストリームリーダをクローズします。

```
result = [];
for i = 1:numAvail
    result = [result read(reader)];
end
close(reader);
```

10．シリアルポートのクローズ

例題は、serialPort オブジェクトについて close を呼び出してシリアルポートをクローズします。

```
close(serialPort);
```

11．入力引数を MATLAB キャラクタ配列に変換

例題の最後のステートメントは、MATLAB 関数 char を使って、配列の入力バイト（整数）をキャラクタの配列に変換します。

```
result = char(result);
```

serialexample プログラムの実行

result の値は、システムの COM1 ポートがオシロスコープに接続しているかどうかにより異なります。オシロスコープを接続して例題を実行した場合は、シリアルポートの読み込みの結果は、以下のようになります。

```
result =
45
```

オシロスコープを接続せずに例題を実行した場合は、読み込むデータはありません。この場合は、空のキャラクタ配列が表示されます。

```
result =
”
```

例題 - Phone Book の作成と利用

例題のメイン関数 phonebook は、電話帳にあるエントリのキーである引数を使わずに、あるいは 1 つ使って呼び出すことができます。関数は、最初に電話帳ファイル用に利用するディレクトリを決定します。

電話帳ファイルが存在しない場合は、java.io.FileOutputStream オブジェクトを作成し、出力ストリームをクローズして作成します。つぎに、ハッシュテーブル内のキーと値の組の格納用の java.util.Hashtable のサブクラスである Java API クラス java.util.Properties のオブジェクトを作成することにより、データ辞書を作成します。phonebookプログラムでは、キーは名前で、値は1つまたは複数の電話番号です。

関数 phonebook は、java.io.FileInputStream オブジェクトを作成することにより、読み込み用の入力ストリームを作成してオープンします。ハッシュテーブルの内容が存在すれば、それをロードするためにオブジェクトについて load を呼び出します。ユーザがルックアップのためにキーをエントリに渡した場合は、エントリを探して表示する pb_lookup を呼び出すことによりエントリをルックアップします。その後、関数 phonebook は、リターンします。

phonebook が名前の引数なしで呼び出された場合は、可能な電話帳のアクションをテキスト化したメニューを表示します。

- エントリのルックアップ
- エントリの追加
- エントリの削除
- エントリ内の電話番号の変更
- すべてのエントリの一覧

メニューは、プログラムを終了する選択肢も含みます。関数は、MATLAB 関数を使ってメニューを表示し、ユーザの選択を入力します。

関数 phonebook は、ユーザがメニューエントリで終了を選択するまで、ユーザの選択を受け取り、要求された電話帳のアクションを繰り返します。その後、関数 phonebook は、java.io.FileOutputStream オブジェクトを作成することにより、ファイル用の出力ストリームをオープンします。オブジェクトについて save を呼び出し、カレントのデータ辞書を電話帳ファイルに書き出します。最後に、出力ストリームをクローズしてリターンします。

関数 phonebook の説明

phonebook が行う主な作業は、以下の通りです。

1．データ辞書とファイル名の決定

最初のステートメントは、電話帳ファイル名 'myphonebook' を変数 pbname に割り当てます。phonebook プログラムが PC 上で実行中の場合は、java.lang.System static メソッド getProperty を呼び出して、データ辞書用に利用するディレクトリを探します。これは、ユーザのカレント作業ディレクトリに設定されます。そうでない場合は、ユーザがあらかじめ任意に定義可能なシステム変数 HOME を使って、ディレクトリを決定するために MATLAB 関数 getenv を使います。その後、pbname をデータ辞書とファイル名 'myphonebook' で構成されるパス名に割り当てます。

```
function phonebook(varargin)
pbname = 'myphonebook'; % データ辞書名
if ispc
  datadir = char(java.lang.System.getProperty('user.dir'));
else
  datadir = getenv('HOME');
end;
pbname = fullfile(datadir, pbname);
```

2．必要ならばファイル出力ストリームを作成

電話帳ファイルが存在しない場合は、phonebook は、ユーザに新規ファイルを作成するかどうかを聞きます。ユーザが y と答えると、phonebook は FileOutputStream オブジェクトを作成することにより新規の電話帳を作成します。try-catch ブロックの try クローズにおいて、FileOutputStream コンストラクタに渡される引数 pbname は、コンストラクタが作成し、オープンするファイル名です。つぎのステートメントは、FileOutputStream ブジェクト FOS について close を呼び出すことによってファイルをクローズします。出力ストリームコンストラクタが失敗すると、catch ステートメントは、メッセージを表示してプログラムを終了します。

```
if ~exist(pbname)
  disp(sprintf('Data file %s does not exist.', pbname));
  r = input('Create a new phone book (y/n)?','s');
```

```
  if r == 'y',
    try
      FOS = java.io.FileOutputStream(pbname);
      FOS.close
    catch
      error(sprintf('Failed to create %s', pbname));
    end;
  else
    return;
  end;
end;
```

3．ハッシュテーブルの作成

例題は、データ辞書用のハッシュテーブルとして java.util.Properties オブジェクトを作成します。

```
pb_htable = java.util.Properties;
```

4．ファイルの入力ストリームの作成

try ブロックにおいて、例題はオブジェクトを FIS に割り当てて、電話帳ファイル名を使って FileInputStream コンストラクタを呼び出します。呼び出しが失敗した場合は、catch ステートメントは、error メッセージを表示してプログラムを終了します。

```
try
  FIS = java.io.FileInputStream(pbname);
catch
  error(sprintf('Failed to open %s for reading.', pbname));
end;
```

5．電話帳のキーと値をロードし、ファイルの入力ストリームをクローズ

例題は、FileInputStream オブジェクト FIS ついて load を呼び出し、電話帳のキーと（存在すれば）その値をハッシュテーブルにロードします。その後ファイルの入力ストリームをクローズします。

```
pb_htable.load(FIS);
FIS.close;
```

6．アクションメニューを表示し、ユーザ選択を取得

while ループ内で、disp ステートメントは、電話帳についてユーザが実行可能なアクションのメニューを表示します。その後、input ステートメントは、ユーザ選択の入力を要求します。

```
while 1
  disp ' '
  disp 'Phonebook Menu:'
  disp ' '
  disp '1. Look up a phone number'
  disp '2. Add an entry to the phone book'
  disp '3. Remove an entry from the phone book'
  disp '4. Change the contents of an entry in the phone book'
  disp '5. Display entire contents of the phone book'
  disp '6. Exit this program'
  disp ' '
  s = input('Please type the number for a menu selection: ','s');
```

7．関数を呼び出し電話帳のアクションを実行

while ループ内で、switch ステートメントは、ユーザの選択に対応する case を与えます。最初の 5 つの case は、関数を呼び出して電話帳のアクションを実行します。

Case 1 は、エントリのキーである名前の入力を指示します。isempty を呼び出してユーザが名前を入力したかどうかを決定します。名前が入力されていない場合は、disp を呼び出してエラーメッセージを表示します。名前が入力された場合は、その名前を pb_lookup に渡します pb_lookup は、エントリをルックアップし、見つかった場合は、エントリの内容を表示します。

```
switch s
  case '1',
    name = input('Enter the name to look up: ','s');
    if isempty(name)
      disp 'No name entered'
    else
      pb_lookup(pb_htable, name);
    end;
```

Case 2 は、pb_add を呼び出して、新規エントリの入力をユーザに指示し、それを電話帳に追加します。

```
case '2',
  pb_add(pb_htable);
```

Case 3 は、input を使って削除するエントリ名の入力を指示します。名前が入力されない場合は、disp を呼び出してエラーメッセージを表示します。名前が入力された場合は、その名前を pb_remove に渡します。

```
case '3',
  name=input('Enter the name of the entry to remove: ', 's');
  if isempty(name)
    disp 'No name entered'
  else
    pb_remove(pb_htable, name);
  end;
```

Case 4 は、input を使って変更するエントリ名の入力を指示します。名前が入力されない場合は、disp を呼び出してエラーメッセージを表示します。名前が入力された場合は、その名前を pb_change に渡します。

```
case '4',
  name=input('Enter the name of the entry to change: ', 's');
  if isempty(name)
    disp 'No name entered'
  else
    pb_change(pb_htable, name);
  end;
```

Case 5 は、pb_listall を呼び出して、全てのエントリを表示します。

```
case '5',
  pb_listall(pb_htable);
```

8．出力ストリームを作成し、電話帳を保存して終了

ユーザが case6 を選択してプログラムを終了する場合は、try ステートメントは FileOuputStream オブジェクトに対するコンストラクタを呼び出し、電話帳名を渡します。コンストラクタが失敗した場合は、catch ステートメントはエラーメッセージを表示します。

オブジェクトが作成された場合は、つぎのステートメントは、FileOutputStream オブジェクト FOS とヘッダ記述文字列を渡して、Properties オブジェクト pb_htable

について save を呼び出すことにより、電話帳データを保存します。その後、FileOutputStream オブジェクトについて、close を呼び出してリターンします。

```
    case '6',
      try
        FOS = java.io.FileOutputStream(pbname);
      catch
        error(sprintf('Failed to open %s for writing.',...
                      pbname));
      end;
      pb_htable.save(FOS,'Data file for phonebook program');
      FOS.close;
      return;
    otherwise
      disp 'That selection is not on the menu.'
    end;
  end;
```

関数 pb_lookup の説明

pb_lookupに渡される引数は、Propertiesオブジェクトpb_htableと要求されるエントリに対する name キーです。関数 pb_lookup は、最初に name キーを使って pb_htable について get を呼び出し、そこでサポート関数 pb_keyfilter が空白をアンダースコアに変更するために呼び出されます。get メソッドは、エントリ (あるいはエントリが見つからない場合は null) を変数 entry に出力します。get は、タイプ java.lang.Object の引数を受け取り、そのタイプの引数を出力することに注意してください。この呼び出しにおいて、get に渡されるキーとそこから出力されるエントリは、実際はキャラクタ配列です。

その後 pb_lookup は isempty を呼び出して、entry が null かどうかを決定します。null の場合は、disp を使って名前が見つからないことを示すメッセージを表示します。entry は null でない場合は、pb_display を呼び出してエントリを表示します。

```
function pb_lookup(pb_htable,name)
entry = pb_htable.get(pb_keyfilter(name));
if isempty(entry),
  disp(sprintf('The name %s is not in the phone book',name));
else
  pb_display(entry);
end
```

関数 pb_add の説明

1．エントリを追加のために入力

関数 pb_add は、1 つの引数、Properties オブジェクト pb_htable を受け取ります。pb_add は、disp を使ってエントリの入力を指示します。上向き矢印 (^) キャラクタをラインデリミタとして使って、input は名前を変数 entry に入力します。その後、 while ループ内で、input を使ってエントリの別の行を変数 line に取り込みます。エントリが終了したことを示す行が空の場合は、コードは while ループから出ます。行が空でない場合は、else ステートメントは、エントリに行を追加し、その後ラインデリミタを追加します。最後に strcmp は input が入力されなかった可能性をチェックし、その場合はリターンします。

```
function pb_add(pb_htable)
disp 'Type the name for the new entry, followed by Enter.'
disp 'Then, type the phone number(s), one per line.'
disp 'To complete the entry, type an extra Enter.'
name = input(':: ','s');
entry=[name '^'];
while 1
  line = input(':: ','s');
  if isempty(line)
    break;
  else
    entry=[entry line '^'];
  end;
end;
if strcmp(entry, '^')
  disp 'No name entered'
  return;
end;
```

2．電話帳にエントリを追加

入力が完了した後で、pb_add は、ハッシュキー name(空白をアンダースコアに変更するために pb_keyfilter が呼び出されます) と entry を使って、pb_htable について put を呼び出します。その後、エントリが追加されたことを示すメッセージが表示されます。

```
pb_htable.put(pb_keyfilter(name),entry);
disp ' '
disp(sprintf('%s has been added to the phone book.', name));
```

関数 pb_remove の説明

1．電話帳でのキーの検索

pb_removeに渡される引数は、Propertiesオブジェクトpb_htableと削除するエントリに対する name キーです。関数 pb_remove は、name キーを使って pb_htable について containsKey を呼び出します。ここでサポート関数 pb_keyfilter が空白をアンダースコアに変更するために呼び出されます。名前が電話帳にない場合は、disp はメッセージを表示し、関数はリターンします。

```
function pb_remove(pb_htable,name)
if ~pb_htable.containsKey(pb_keyfilter(name))
  disp(sprintf('The name %s is not in the phone book',name))
  return
end;
```

2．確認してキーを削除

キーがハッシュテーブルにある場合は、pb_remove はユーザに確認を求めます。ユーザが y を入力して削除を確認すると、(フィルタされた)name キーを使ってpb_removeはpb_htableについてremoveを呼び出し、エントリが削除されたことを示すメッセージを表示します。ユーザが n を入力した場合は、削除は行われず、disp は削除が行われなかったことを示すメッセージを表示します。

```
r = input(sprintf('Remove entry %s (y/n)? ',name), 's');
if r == 'y'
  pb_htable.remove(pb_keyfilter(name));
  disp(sprintf('%s has been removed from the phone book',name))
else
  disp(sprintf('%s has not been removed',name))
end;
```

関数 pb_change の説明

1．変更するエントリを見つけて確認

pb_changeに渡される引数は、Propertiesオブジェクトpb_htableと要求されるエントリに対する name キーです。関数 pb_change は、name キーを使って pb_htable について get を呼び出します。ここで、空白をアンダースコアに変更するために pb_keyfilter が呼ばれます。get メソッドは、エントリ (あるいはエントリが見つからない場合は null) を変数 entry に出力します。pb_change は、isempty を呼び出してエントリが空かどうかを決定します。エントリが空の場合は、pb_change は名前が

電話帳に追加されるというメッセージを表示し、ユーザはエントリに対して電話番号を入力することができます。

else文でエントリが見つかった場合は、pb_changeはpb_displayを呼び出してエントリを表示します。その後、input を使って代わりのエントリを確認します。ユーザが y 以外を入力した場合は、関数はリターンします。

```
function pb_change(pb_htable,name)
entry = pb_htable.get(pb_keyfilter(name));
if isempty(entry)
  disp(sprintf('The name %s is not in the phone book', name));
  return;
else
  pb_display(entry);
  r = input('Replace phone numbers in this entry (y/n)? ','s');
  if r ~= 'y'
    return;
  end;
end;
```

2．新規の電話番号を入力し電話帳のエントリを変更

pb_change は、disp を使って新規の電話番号の入力のためのプロンプトを表示します。その後 pb_change は、「エントリを追加のために入力」で説明したのと同じステートメントを使って、データを変数 entry に入力します。

その後、既存のエントリと新規のエントリを置き換えるために、pb_change は（フィルタされた）キー name と新規エントリを使って pb_htable について put を呼び出します。その後で、エントリが変更されたことを示すメッセージを表示します。

```
disp 'Type in the new phone number(s), one per line.'
disp 'To complete the entry, type an extra Enter.'
disp(sprintf(':: %s', name));
entry=[name '^'];
while 1
  line = input(':: ','s');
  if isempty(line)
    break;
  else
    entry=[entry line '^'];
  end;
```

```
end;
pb_htable.put(pb_keyfilter(name),entry);
disp ' '
disp(sprintf('The entry for %s has been changed', name));
```

関数 pb_listall の説明

関数 pb_listall は、1 つの引数、Properties オブジェクト pb_htable を受け取ります。関数は、pb_htable オブジェクトについて propertyNames を呼び出し、enum に java.util.Enumeration オブジェクトを出力します。これは、全てのキーの例示をサポートします。while ループ内で、pb_listall は enum について hasMoreElements を呼び出して、真を出力する場合は pb_listall は enum について nextElement を呼び出してつぎのキーを出力します。その後 pb_display を呼び出してキーとエントリを表示します。これはキーを使って pb_htable について get を呼び出すことにより取得します。

```
function pb_listall(pb_htable)
enum = pb_htable.propertyNames;
while enum.hasMoreElements
  key = enum.nextElement;
  pb_display(pb_htable.get(key));
end;
```

関数 pb_display の説明

関数 pb_display は、電話帳のエントリである引数 entry を受け取ります。水平線を表示した後で、pb_display は MATLAB 関数 strtok を呼び出して最初の行のラインデリミタ (^) までを t に、残りを r に取り込みます。その後、t が空のときに終了する while ループ内で、t のカレント行を表示します。その後、strtok を呼び出してつぎの行のrからをtに取り込みます。すべてのラインが表示されると、pb_display は、別の水平線を表示してエントリの終わりを示します。

```
function pb_display(entry)
disp ' '
disp '-------------------------'
[t,r] = strtok(entry,'^');
while ~isempty(t)
  disp(sprintf(' %s',t));
  [t,r] = strtok(r,'^');
end;
disp '-------------------------'
```

関数 pb_keyfilter の説明

関数 pb_keyfilter は、ハッシュテーブルのキーとして利用される名前である引数 key を受け取り、格納するためにフィルタするか、あるいは表示用にフィルタ処理をしません。キー内の空白をアンダースコア (_) で置き換えるフィルタは、java.util.Properties のメソッドを使ってキーを利用可能にします。

```
function out = pb_keyfilter(key)
if ~isempty(findstr(key,' '))
  out = strrep(key,' ','_');
else
  out = strrep(key,'_',' ');
end;
```

phonebook プログラムの実行

この実行例においては、ユーザは、引数を使わずに phonebook を呼び出します。現在電話帳にある 2 つのエントリを表示するメニューのアクション 5 を選択します (すべてのエントリがフィクションです)。それから、エントリを追加するために 2 を選択します。エントリを追加した後で、再度 5 を選択して新規エントリと共に 2 個のエントリを表示します。

```
Phonebook Menu:

  1. Look up a phone number
  2. Add an entry to the phone book
  3. Remove an entry from the phone book
  4. Change the contents of an entry in the phone book
  5. Display entire contents of the phone book
  6. Exit this program

Please type the number for a menu selection: 5

-------------------------
 Sylvia Woodland
 (508) 111-3456
-------------------------

-------------------------
 Russell Reddy
 (617) 999-8765
-------------------------
```

```
Phonebook Menu:

    1. Look up a phone number
    2. Add an entry to the phone book
    3. Remove an entry from the phone book
    4. Change the contents of an entry in the phone book
    5. Display entire contents of the phone book
    6. Exit this program

Please type the number for a menu selection: 2

Type the name for the new entry, followed by Enter.
Then, type the phone number(s), one per line.
To complete the entry, type an extra Enter.
:: BriteLites Books
:: (781) 777-6868
::

BriteLites Books has been added to the phone book.

Phonebook Menu:

    1. Look up a phone number
    2. Add an entry to the phone book
    3. Remove an entry from the phone book
    4. Change the contents of an entry in the phone book
    5. Display entire contents of the phone book
    6. Exit this program

Please type the number for a menu selection: 5

-------------------------
 BriteLites Books
 (781) 777-6868
-------------------------


-------------------------
 Sylvia Woodland
```

```
(508) 111-3456
-------------------------

-------------------------
Russell Reddy
(617) 999-8765
-------------------------
```

6

データの読み込みと書き込み

MATLAB 環境とのデータのやりとりを行うために、データをディスクに保存するためのデータファイルフォーマットである MAT- ファイルを利用できます。MAT- ファイルは、異なるプラットフォーム間で非常にポータブルな方法で MATLAB データを移動するための便利なメカニズムを提供します。さらに、他のスタンドアロン MATLAB アプリケーションとのデータのやりとりの方法を提供します。

MATLAB 外部のアプリケーションで MAT- ファイルを簡単に利用するために、MAT- ファイルの読み書きを行う C または Fortran プログラム内で利用するアクセスルーチンのライブラリを提供しています。MAT- ファイルにアクセスするプログラムも、本章で説明する mx API ルーチンを利用します。

つぎの一覧は、本章の内容をまとめたものです。

- MAT- ファイルの使用法
- MAT- ファイルの例
- MAT- ファイルプログラムのコンパイルとリンク

ユーザアプリケーションの作成における情報とサポートについては、「その他の情報」を参照してください。

MAT- ファイルの使用法

本節は、MATLAB 環境とのデータのやりとりを行うための様々な手法を説明します。説明する主なトピックスは、以下の通りです。

- MATLAB のデータの読み込み
- MATLAB からのデータの書き出し
- プラットフォーム間のデータファイルの交換
- MAT- ファイルの読み込みと書き出し
- 関連するファイル

データのやりとりの最も重要な方法は、データをディスクに保存するためのデータファイルフォーマットである MAT- ファイルの利用に関連します。MAT- ファイルは、異なるプラットフォーム間での MATLAB データの移動や、他のスタンドアロン MATLAB アプリケーションとのデータのやりとりに便利なメカニズムを提供します。

MATLAB 外部のアプリケーションで MAT- ファイルを簡単に利用するために、MAT-ファイルの読み書きを行うCまたはFortranプログラム内で利用する、接頭語 mat の付いたアクセスルーチンのライブラリを開発しました。MAT- ファイルにアクセスするプログラムも、「C 言語 MEX- ファイルの作成」および「Fortran MEX-ファイルの作成」で説明する接頭語 mx の付いた API ルーチンを利用します。

MATLAB へのデータの読み込み

いくつかの手法により MATLAB にプログラムを読み込むことができます。最も良い読み込み方法は、データ量、データがマシンが読める形式になっているかどうか、データの形式に依存します。いくつかの選択から、要求に最も合うものを選択してください。

- **データを明示的な要素のリストとして入力**

  要素が 10 から 15 以下でデータ量が小さければ、鍵括弧 [ ] を使って明示的にデータを簡単にタイプできます。この手法は、間違えた場合に入力を修正できないので、大量のデータは扱いにくいです。

- **M- ファイル内でデータを作成**

  明示的な要素のリストとしてデータを入力する M- ファイルを作成するためにテキストエディタを使います。この手法は、データがコンピュータが読めない形式で、ユーザがタイプしなければならないときに有効です。最初の方法と本質的に同じですが、この手法はデータを変更し、間違いを直すためにエディタ

を使うことができる利点をもちます。データを再入力するために M- ファイルを再実行するだけで行うことができます。

- **ASCII フラットファイルからデータをロード**

  *フラットファイル*は、固定長の行の最後がニューライン (キャリッジリターン) で、スペースで数値を区切る ASCII 形式でデータを保存します。標準のテキストエディタを使って ASCII フラットファイルを編集できます。フラットファイルは、load コマンドを使って直接 MATLAB に読み込めます。結果として、ファイル名と同じ名前の変数を作成します。

- **MATLAB の I/O 関数を使ってデータを読み込み**

  fopen, fread および MATLAB のその他の低レベル入出力関数を使って、データを読み込みます。この手法は、決まったファイル形式をもつ他のアプリケーションからデータファイルをロードするのに役立ちます。

- **データを読み込む MEX- ファイルを作成**

  サブルーチンが、他のアプリケーションからデータファイルを読むことが既に可能である場合に、選択すべき方法です。詳細は、「MEX- ファイルの紹介」を参照してください。

- **ユーザデータを変換してプログラムを作成**

  ユーザデータを MAT- ファイル形式に変換するためのプログラムを C または Fortran で書くことができます。その後、load コマンドを使って MAT- ファイルを MATLAB に読み込みます。詳細は、「MAT- ファイルの読み込みと書き出し」を参照してください。

MATLAB からのデータの書き出し

MATLAB データを外部に戻す数種の方法があります。

- **ダイアリファイルの作成**

  小さい行列に対しては、diary コマンドを使ってダイアリファイルを作成し、このファイルに変数を表示してファイルに書き込みます。後でダイアリファイルを編集するためにテキストエディタを使うことができます。diary の出力は、セッション中に使う MATLAB コマンドを含み、ドキュメントやレポートへ挿入するのに役立ちます。

- **Save コマンドの利用**

  -ascii オプションを指定して save コマンドを使って、ASCII 形式でデータを保存します。たとえば、

  ```
  A = rand(4,3);
  save temp.dat A -ascii
  ```

  は、つぎのような temp.dat という ASCII ファイルを作成します。

  ```
  1.3889088e-001  2.7218792e-001  4.4509643e-001
  2.0276522e-001  1.9881427e-001  9.3181458e-001
  1.9872174e-001  1.5273927e-002  4.6599434e-001
  6.0379248e-001  7.4678568e-001  4.1864947e-001
  ```

  -ascii オプションは、数値行列形式のデータのみをサポートします。数値配列 (2 次元以上)、セル配列、構造体はサポートされません。

  詳細は、save 関数リファレンスを参照してください。

- **MATLAB I/O 関数の利用**

  fopen, fwrite およびその他の低レベル入出力関数を使って、特殊なフォーマットでデータを書き出します。この手法は、他のアプリケーションで要求されるファイル形式でデータファイルを書き出す場合に役立ちます。

- **MEX- ファイルを作成してデータを書き出す**

  データを書き出すための MEX-ファイルを作成することができます。サブルーチンが、他のアプリケーションで必要な形式でデータファイルを書き出すことが既に可能である場合は、選択すべき方法です。詳細は、「MEX- ファイルの紹介」を参照してください。

- **MAT- ファイルからデータを変換**

  saveコマンドを使ってデータをMAT-ファイルとして書き出すことができます。その後、ユーザ固有の形式に MAT- ファイルを変換するためのプログラムを C または Fortran で作成することができます。詳細は、「MAT- ファイルの読み込みと書き出し」を参照してください。

プラットフォーム間でのデータファイルの交換

数種の異なるコンピュータシステムで MATLAB インプリメンテーションを行ったり、他のシステム上のユーザに MATLAB アプリケーションを送信する必要がある場合があります。MATLAB アプリケーションは、関数とスクリプトを含む M- ファイルとバイナリデータを含む MAT- ファイルから成ります。

M- ファイルはプラットフォームに依存せず、MAT- ファイルのヘッダにマシンのサインがあるため、両方ともマシン間で直接送信することができます。MATLAB は、ファイルをロードするときにサインをチェックし、サインがファイルが外部のものであることを示せば必要な変換を行います。

異なるマシンアーキテクチャで MATLAB を使うことは、様々なマシン間でバイナリデータと ASCII データの変換機能を必要とします。このタイプの機能の例は、FTP, NFS, Kermit その他の通信プログラムです。これらのプログラムを使うとき、バイナリ MAT- ファイルはバイナリファイルモード で、ASCII M- ファイルは *ASCII ファイルモード* で送信することに注意してください。モードを正しく設定しないと、データが壊れます。

MAT- ファイルの読み込みと書き出し

MATLAB の save コマンドは、カレントのメモリにある MATLAB 配列を MAT- ファイルと呼ばれるバイナリディスクファイルに保存します。これらのファイルは、.mat という拡張子をもつため、MAT- ファイルという用語が使われます。load コマンドは、逆の操作を行います。load コマンドは、MATLAB のワークスペースにディスク上の MAT- ファイルから MATLAB 配列を読み込みます。

MAT- ファイルは、MATLAB 5 以降でサポートされているデータタイプの 1 つまたは複数を含みます。データタイプは、文字列、行列、多次元配列、構造体、セル配列です。MATLAB は、連続したバイトストリームとしてディスクに順番にデータを書き出します。

MAT- ファイルインタフェースライブラリ

MAT- ファイルインタフェースライブラリは、MAT- ファイルの読み込みと書き出しのためのサブルーチンを含みます。ユーザの C や Fortran プログラム内からこれらのルーチンを呼び出せます。この操作を行うためには、ユーザがコードを書くよりも、これらのルーチンを使うことを推奨します。ライブラリ内のルーチンを使うと、MAT- ファイルの構造の将来的な変化の影響を受けません。

MAT- ファイルライブラリは、MAT- ファイルの読み込みと書き出しのためのルーチンを含みます。これらは、すべて 3 文字の接頭語 mat から始まります。つぎの表は、利用可能な MAT- ファンクションとその目的を示したものです。

表 6-1: C MAT- ファイルルーチン

MAT-Function	目的
matOpen	MAT- ファイルをオープンします。
matClose	MAT- ファイルをクローズします。
matGetDir	MAT- ファイルから MATLAB 配列のリストを取得します。
matGetFp	MAT- ファイルの ANSI C ファイルのポインタを取得します。
matGetArray	MAT- ファイルから MATLAB 配列を読み込みます。
matPutArray	MAT-ファイルにMATLAB配列を書き出します。
matGetNextArray	MAT- ファイルからつぎの MATLAB 配列を読み込みます。
matDeleteArray	MAT-ファイルからMATLAB配列を削除します。
matPutArrayAsGlobal	load コマンドがグローバルワークスペースに MATLAB 配列を置くように、MAT- ファイルに MATLAB 配列を置きます。
matGetArrayHeader	MAT- ファイル (データをもたない) から MATLAB 配列のヘッダをロードします。
matGetNextArrayHeader	MAT- ファイル (データをもたない) からつぎの MATLAB 配列のヘッダをロードします。

表 6-2: Fortran MAT- ファイルルーチン

MAT-Function	目的
matOpen	MAT- ファイルをオープンします。
matClose	MAT- ファイルをクローズします。
matGetDir	MAT- ファイルから MATLAB 配列のリストを取得します。
matGetMatrix	MAT- ファイルから名前の付いた MATLAB 配列を取得します。
matPutMatrix	MAT- ファイルに MATLAB 配列を設定します。
matGetNextMatrix	MAT- ファイルからつぎの連続する MATLAB 配列を取得します。
matDeleteMatrix	MAT- ファイルから MATLAB 配列を削除します。
matGetString	MAT- ファイルから MATLAB 文字列を読み込みます。
matPutString	MAT- ファイルに MATLAB 文字列を書き出します。

関連するファイル

MAT- ファイルの読み込みと書き出しに関連するファイルは、ディスク上にあります。つぎの表「MAT-Function サブディレクトリ」は、MAT- ファンクションを

使ったデータの読み書きのために必要なサブディレクトリのパスを示したものです。

表 6-3: MAT- ファンクションサブディレクトリ

プラットフォーム	内容	ディレクトリ
Windows	インクルードファイル	<matlab>\extern\include
	ライブラリ	<matlab>\bin\win32
	例題	<matlab>\extern\examples/eng_mat
UNIX	インクルードファイル	<matlab>/extern/include
	ライブラリ	<matlab>/extern/lib/$arch
	例題	<matlab>/extern/examples/eng_mat

インクルードファイル

include ディレクトリには、API ライブラリでアクセスできるルーチンに対するプロトタイプをもつ関数の宣言を含むヘッダファイルがあります。サブディレクトリに含まれるのはつぎのものです。

- matrix.h は、MATLAB 配列のアクセスと作成法を定義するヘッダファイルです。
- mat.h は、MAT- ファイルのアクセスと作成法を定義するヘッダファイルです。

ライブラリ

ユーザプログラムのリンクのための共有 (ダイナミックリンク) ライブラリを含むサブディレクトリは、プラットフォームに依存します。

Windows の共有ライブラリ

bin サブディレクトリは、ユーザプログラムのリンクのための共有ライブラリを含みます。

- libmat.dll, MAT- ファイルルーチン (C および Fortran) のライブラリ
- libmx.dll, 配列のアクセスと作成ルーチンのライブラリ

UNIX の共有ライブラリ

extern/lib/$arch サブディレクトリは、ユーザプログラムのリンクのための共有ライブラリを含みます。$arch はユーザマシンのアーキテクチャです。たとえば、sol2 では、サブディレクトリは、extern/lib/sol2 です。

- libmat.so, MAT- ファイルルーチン (C および Fortran) のライブラリ
- libmx.so, 配列のアクセスと作成ルーチンのライブラリ

例題ファイル

examples/eng_mat サブディレクトリは、MAT- ファイルルーチンの使用法を説明する例題ファイルの C および Fortran ソースコードを含みます。ソースコードファイルは、Windows と UNIX とで同じです。

表 6-4: C および Fortran の例題

ライブラリ	説明
matcreat.c	MATLAB へのロードが可能な MAT- ファイルを作成するライブラリルーチンの使用法を説明する例題の C プログラム
matdgns.c	MAT- ファイルの読み込みと診断を行うライブラリルーチンの使用法を説明する例題の C プログラム
matdemo1.f	Fortran プログラムから MATLAB MAT- ファイルファンクションの呼び出し方法を説明する例題の Fortran プログラム
matdemo2.f	matdemo1.f によって作成された MAT- ファイルに読み込み、その内容を記述するライブラリルーチンの使用法を説明する例題の Fortran プログラム

MATLAB API のファイルとディレクトリに関する情報は、「その他の情報」を参照してください。

MAT- ファイルの例

この節では、MAT- ファイルの書き出し、読み込み、診断の C および Fortran の例題を説明します。含まれるトピックスは以下の通りです。

- MAT- ファイルを C で作成
- MAT- ファイルを C で読み込む
- MAT- ファイルを Fortran で作成
- MAT- ファイルを Fortran で読み込む

MAT- ファイルを C で作成

この例題プログラムは、MATLAB にロードされる MAT- ファイルを作成するライブラリルーチンの使用法を記述しています。プログラムは、読み込みまたは書き出しの失敗に対する MAT- ファンクション呼び出しの返り値のチェック方法も示しています。

```
/*
 * MAT-file creation program
 *
 * See the MATLAB API Guide for compiling information.
 *
 * Calling syntax:
 *
 *   matcreat
 *
 * Create a MAT-file which can be loaded into MATLAB.
 *
 * This program demonstrates the use of the following functions:
 *
 *  matClose
 *  matGetArray
 *  matOpen
 *  matPutArray
 *  matPutArrayAsGlobal
 *
 * Copyright 1984-2000 The MathWorks, Inc.
 */
/* $Revision: 1.10 $ */
#include <stdio.h>
```

```
#include <string.h> /* For strcmp() */
#include <stdlib.h> /* For EXIT_FAILURE, EXIT_SUCCESS */
#include "mat.h"

#define BUFSIZE 256

int main() {
  MATFile *pmat;
  mxArray *pa1, *pa2, *pa3;
  double data[9] = { 1.0, 4.0, 7.0, 2.0, 5.0, 8.0, 3.0, 6.0, 9.0 };
  const char *file = "mattest.mat";
  char str[BUFSIZE];
  int status;

  printf("Creating file %s...\n\n", file);
  pmat = matOpen(file, "w");
  if (pmat == NULL) {
    printf("Error creating file %s\n", file);
    printf("(Do you have write permission in this directory?)\n");
    return(EXIT_FAILURE);
  }

  pa1 = mxCreateDoubleMatrix(3,3,mxREAL);
  if (pa1 == NULL) {
    printf("%s : Out of memory on line %d\n",
           __FILE__, __LINE__);
    printf("Unable to create mxArray.\n");
    return(EXIT_FAILURE);
  }
  mxSetName(pa1, "LocalDouble");

  pa2 = mxCreateDoubleMatrix(3,3,mxREAL);
  if (pa2 == NULL) {
    printf("%s : Out of memory on line %d\n",
           __FILE__, __LINE__);
    printf("Unable to create mxArray.\n");
    return(EXIT_FAILURE);
  }
  mxSetName(pa2, "GlobalDouble");
  memcpy((void *)(mxGetPr(pa2)), (void *)data, sizeof(data));
```

```
pa3 = mxCreateString("MATLAB: the language of technical
                     computing");
if (pa3 == NULL) {
  printf("%s :  Out of memory on line %d\n",
         __FILE__, __LINE__);
  printf("Unable to create string mxArray.\n");
  return(EXIT_FAILURE);
}
mxSetName(pa3, "LocalString");

status = matPutArray(pmat, pa1);
if (status != 0) {
  printf("%s :  Error using matPutArray on line %d\n",
         __FILE__, __LINE__);
  return(EXIT_FAILURE);
}

status = matPutArrayAsGlobal(pmat, pa2);
if (status != 0) {
  printf("Error using matPutArrayAsGlobal\n");
  return(EXIT_FAILURE);
}

status = matPutArray(pmat, pa3);
if (status != 0) {
  printf("%s :  Error using matPutArray on line %d\n",
         __FILE__, __LINE__);
  return(EXIT_FAILURE);
}

/*
 * Ooops! we need to copy data before writing the array.  (Well,
 * ok, this was really intentional.) This demonstrates that
 * matPutArray will overwrite an existing array in a MAT-file.
 */
memcpy((void *)(mxGetPr(pa1)), (void *)data, sizeof(data));
status = matPutArray(pmat, pa1);
```

```
if (status != 0) {
  printf("%s :  Error using matPutArray on line %d\n",
         __FILE__, __LINE__);
  return(EXIT_FAILURE);
}

/* clean up */
mxDestroyArray(pa1);
mxDestroyArray(pa2);
mxDestroyArray(pa3);

if (matClose(pmat) != 0) {
  printf("Error closing file %s\n",file);
  return(EXIT_FAILURE);
}

/*
 * Reopen file and verify its contents with matGetArray
 */
pmat = matOpen(file, "r");
if (pmat == NULL) {
  printf("Error reopening file %s\n", file);
  return(EXIT_FAILURE);
}

/*
 * Read in each array we just wrote
 */
pa1 = matGetArray(pmat, "LocalDouble");
if (pa1 == NULL) {
  printf("Error reading existing matrix LocalDouble\n");
  return(EXIT_FAILURE);
}
if (mxGetNumberOfDimensions(pa1) != 2) {
  printf("Error saving matrix: result does not have two
         dimensions\n");
  return(EXIT_FAILURE);
}
```

```
  pa2 = matGetArray(pmat, "GlobalDouble");
  if (pa2 == NULL) {
    printf("Error reading existing matrix GlobalDouble\n");
    return(EXIT_FAILURE);
  }
  if (!(mxIsFromGlobalWS(pa2))) {
    printf("Error saving global matrix: result is not global\n");
    return(EXIT_FAILURE);
  }

  pa3 = matGetArray(pmat, "LocalString");
  if (pa3 == NULL) {
    printf("Error reading existing matrix LocalDouble\n");
    return(EXIT_FAILURE);
  }

  status = mxGetString(pa3, str, sizeof(str));
  if(status != 0) {
    printf("Not enough space. String is truncated.");
    return(EXIT_FAILURE);
  }
  if (strcmp(str, "MATLAB: the language of technical
                computing")) {
    printf("Error saving string: result has incorrect
          contents\n");
    return(EXIT_FAILURE);
  }

  /* clean up before exit */
  mxDestroyArray(pa1);
  mxDestroyArray(pa2);
  mxDestroyArray(pa3);

  if (matClose(pmat) != 0) {
    printf("Error closing file %s\n",file);
    return(EXIT_FAILURE);
  }
  printf("Done\n");
  return(EXIT_SUCCESS);
}
```

この例題プログラムの実行可能なバージョンを作成するには、ファイルをコンパイルし適切なライブラリをリンクしてください。種々のプラットフォームでの MAT- ファイルプログラムのコンパイルとリンクの方法の詳細は、「MAT- ファイルプログラムのコンパイルとリンク」に記述されています。

MAT- ファイルプログラムをコンパイルしリンクすると、作成したスタンドアロンアプリケーションを実行できます。このプログラムは、MATLAB にロード可能な MAT- ファイル mattest.mat を作成します。アプリケーションを実行するには、プラットフォームによって、アイコンをダブルクリックするか、システムプロンプトで matcreat と入力してください。

```
matcreat
Creating file mattest.mat...
```

MAT- ファイルが作成されたことを確認するには、MATLAB プロンプトで、つぎを入力します。

```
whos -file mattest.mat
  Name            Size        Bytes  Class

  GlobalDouble     3x3           72  double array (global)
  LocalDouble      3x3           72  double array
  LocalString      1x43          86  char array

Grand total is 61 elements using 230 bytes
```

MAT- ファイルを C で読み込む

この例題プログラムは、MAT- ファイルの読み込みや診断を行うライブラリルーチンの使用法を記述します。

```
/* $Revision: 1.1 $ */
/*
 * MAT-file diagnose program
 *
 * Calling syntax:
 *
 *   matdgns <matfile.mat>
 *
 * It diagnoses the MAT-file named <matfile.mat>.
 *
 * This program demonstrates the use of the following functions:
```

```
 *
 *  matClose
 *  matGetDir
 *  matGetNextArray
 *  matGetNextArrayHeader
 *  matOpen
 *
 *  Copyright (c) 1984-1998 The MathWorks, Inc.
 */

#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "mat.h"

int diagnose(const char *file) {
  MATFile*pmat;
  char**dir;
  int ndir;
  int i;
  mxArray *pa;

  printf("Reading file %s...\n\n", file);

  /* Open file to get directory. */
  pmat = matOpen(file, "r");
  if (pmat == NULL) {
    printf("Error opening file %s\n", file);
    return(1);
  }

  /* Get directory of MAT-file. */
  dir = matGetDir(pmat, &ndir);
  if (dir == NULL) {
    printf("Error reading directory of file %s\n", file);
    return(1);
  } else {
    printf("Directory of %s:\n", file);
    for (i=0; i < ndir; i++)
      printf("%s\n",dir[i]);
```

```
}
mxFree(dir);

/* In order to use matGetNextXXX correctly, reopen file to read
   in headers. */
if (matClose(pmat) != 0) {
  printf("Error closing file %s\n",file);
  return(1);
}
pmat = matOpen(file, "r");
if (pmat == NULL) {
  printf("Error reopening file %s\n", file);
  return(1);
}

/* Get headers of all variables. */
printf("\nExamining the header for each variable:\n");
for (i=0; i < ndir; i++) {
  pa = matGetNextArrayHeader(pmat);
  if (pa == NULL) {
    printf("Error reading in file %s\n", file);
    return(1);
  }

  /* Diagnose header pa. */
  printf("According to its header, array %s has %d
         dimensions\n", mxGetName(pa),
         mxGetNumberOfDimensions(pa));
  if (mxIsFromGlobalWS(pa))
    printf("  and was a global variable when saved\n");
  else
    printf("  and was a local variable when saved\n");
  mxDestroyArray(pa);
}

/* Reopen file to read in actual arrays. */
if (matClose(pmat) != 0) {
  printf("Error closing file %s\n",file);
  return(1);
}
```

```
  pmat = matOpen(file, "r");
  if (pmat == NULL) {
    printf("Error reopening file %s\n", file);
    return(1);
  }

  /* Read in each array. */
  printf("\nReading in the actual array contents:\n");
  for (i=0; i<ndir; i++) {
    pa = matGetNextArray(pmat);
    if (pa == NULL) {
      printf("Error reading in file %s\n", file);
      return(1);
    }

    /* Diagnose array pa. */
    printf("According to its contents, array %s has %d
           dimensions\n", mxGetName(pa),
           mxGetNumberOfDimensions(pa));
    if (mxIsFromGlobalWS(pa))
      printf("  and was a global variable when saved\n");
    else
      printf("  and was a local variable when saved\n");
    mxDestroyArray(pa);
  }

  if (matClose(pmat) != 0) {
    printf("Error closing file %s\n",file);
    return(1);
  }
  printf("Done\n");
  return(0);
}

int main(int argc, char **argv)
{
  int result;

  if (argc > 1)
    result = diagnose(argv[1]);
```

```
  else {
    result = 0;
    printf("Usage: matdgns <matfile>");
    printf("where <matfile> is the name of the MAT-file");
    printf("to be diagnosed");
  }

  return (result==0)?EXIT_SUCCESS:EXIT_FAILURE;
}
```

このプログラムをコンパイル、リンクした後で、結果を見ることができます。

```
matdgns mattest.mat
Reading file mattest.mat...

Directory of mattest.mat:
GlobalDouble
LocalString
LocalDouble

Examining the header for each variable:
According to its header, array GlobalDouble has 2 dimensions
  and was a global variable when saved
According to its header, array LocalString has 2 dimensions
  and was a local variable when saved
According to its header, array LocalDouble has 2 dimensions
  and was a local variable when saved

Reading in the actual array contents:
According to its contents, array GlobalDouble has 2 dimensions
  and was a global variable when saved
According to its contents, array LocalString has 2 dimensions
  and was a local variable when saved
According to its contents, array LocalDouble has 2 dimensions
  and was a local variable when saved
Done
```

MAT- ファイルを Fortran で作成

この例題は、MAT- ファイル matdemo.mat を作成します。

```
C $Revision: 1.6 $
C
C    matdemo1.f
C
C    This is a simple program that illustrates how to call the
C    MATLAB MAT-file functions from a Fortran program.  This
C    demonstration focuses on writing MAT-files.
C
C    Copyright (c) 1984-2000 The MathWorks, Inc.
C
C


C
C    matdemo1 - Create a new MAT-file from scratch.
C
     program matdemo1
C----------------------------------------------------------------
C    (integer) Replace integer by integer*8 on the DEC Alpha
C    platform.
C
     integer matOpen, mxCreateFull, mxCreateString
     integer matGetMatrix, mxGetPr
     integer mp, pa1, pa2, pa3
C----------------------------------------------------------------
C
C    Other variable declarations here
C
     integer status, matClose
     double precision dat(9)
     data dat / 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0 /
C
C    Open MAT-file for writing.
C
     write(6,*) 'Creating MAT-file matdemo.mat ...'
     mp = matOpen('matdemo.mat', 'w')
```

```
      if (mp .eq. 0) then

     write(6,*) 'Can't open "matdemo.mat" for writing.'
     write(6,*) '(Do you have write permission in this directory?)'

        stop
      end if
C
C     Create variables.
C
      pa1 = mxCreateFull(3,3,0)
      call mxSetName(pa1, 'Numeric')
C
      pa2 = mxCreateString('MATLAB: The language of computing')
      call mxSetName(pa2, 'String')
C
      pa3 = mxCreateString('MATLAB: The language of computing')
      call mxSetName(pa3, 'String2')
C
      call matPutMatrix(mp, pa1)
      call matPutMatrix(mp, pa2)
      call matPutMatrix(mp, pa3)
C
C     Whoops! Forgot to copy the data into the first matrix --
C     it's now blank. (Well, ok, this was deliberate.)  This
C     demonstrates that matPutMatrix will overwrite existing
C     matrices.
C
      call mxCopyReal8ToPtr(dat, mxGetPr(pa1), 9)
      call matPutMatrix(mp, pa1)
C
C     Now, we'll delete String2 from the MAT-file.
C
      call matDeleteMatrix(mp, 'String2')
C
C     Finally, read back in MAT-file to make sure we know what we
C     put into it.
C
      status = matClose(mp)
```

```
      if (status .ne. 0) then
        write(6,*) 'Error closing MAT-file'
        stop
      end if
C
      mp = matOpen('matdemo.mat', 'r')
      if (status .ne. 0) then
        write(6,*) 'Can''t open ''matdemo.mat'' for reading.'
        stop
      end if
C
      pa1 = matGetMatrix(mp, 'Numeric')
      if (mxIsNumeric(pa1) .eq. 0) then
       write(6,*) 'Invalid non-numeric matrix written to MAT-file'
       stop
      end if
C
      pa2 = matGetMatrix(mp, 'String')
      if (mxIsString(pa2) .eq. 0) then
       write(6,*) 'Invalid non-numeric matrix written to MAT-file'
       stop
      end if
C
      pa3 = matGetMatrix(mp, 'String2')
      if (pa3 .ne. 0) then
        write(6,*) 'String2 not deleted from MAT-file'
        stop
      end if
C
      status = matClose(mp)
      if (status .ne. 0) then
        write(6,*) 'Error closing MAT-file'
        stop
      end if
C
      write(6,*) 'Done creating MAT-file'
      stop
      end
```

MAT- ファイルプログラムをコンパイルし、リンクすると、作成したスタンドアロンアプリケーションを実行できます。このプログラムは、MATLAB にロード可能な MAT- ファイル matdemo.mat を作成します。アプリケーションを実行するには、プラットフォームによって、アイコンをダブルクリックするか、システムプロンプトで matdemo1 と入力してください。

```
matdemo1
Creating MAT-file matdemo.mat ...
Done creating MAT-file
```

MAT- ファイルが作成されたことを確認するには、MATLAB プロンプトでつぎを入力します。

```
whos -file matdemo.mat
Name         Size        Bytes  Class

Numeric      3x3            72  double array
String       1x33           66  char array

Grand total is 42 elements using 138 bytes
```

注意　Windows スタンドアロンプログラム（MAT- ファイル固有でない）の例題は、<matlab>\extern\examples\eng_mat ディレクトリの engwindemo.c を参照してください。

MAT- ファイルを Fortran で読み込む

この例題プログラムは、matdemo1.f で作成された MAT- ファイルに読み込み、その内容を記述するライブラリルーチンの使用法を説明します。

```
C     matdemo2.f
C
C     This is a simple program that illustrates how to call the
C     MATLAB MAT-file functions from a Fortran program.  This
C     demonstration focuses on reading MAT-files. It reads in
C     the MAT-file created by matdemo1.f and describes its
C     contents.
C
C     Copyright (c) 1996-2000 The MathWorks, Inc.
```

```
C
C--------------------------------------------------------------
C     $Revision: 1.7 $
C
      program matdemo2
C--------------------------------------------------------------
C     (integer) Replace integer by integer*8 on the DEC Alpha
C     platform.
C
      integer matOpen, matGetDir, matGetNextMatrix
      integer mp, dir, adir(100), pa
C--------------------------------------------------------------
C
C     Other variable declarations here
C
      integer   mxGetM, mxGetN, matClose
      integer   ndir, i, stat
      character*32 names(100), name, mxGetName
C
C--------------------------------------------------------------
C     Open file and read directory.
C--------------------------------------------------------------
C
      mp = matOpen('matdemo.mat', 'r')
      if (mp .eq. 0) then
        write(6,*) 'Can't open "matdemo.mat".'
        stop
      end if
C
C     Read directory.
C
      dir = matgetdir(mp, ndir)
      if (dir .eq. 0) then
        write(6,*) 'Can't read directory.'
        stop
      endif
C
C     Copy integer into an array of pointers.
C
      call mxCopyPtrToPtrArray(dir, adir, ndir)
```

```
C
C     Copy integer to character string
C
      do 20 i=1,ndir
        call mxCopyPtrToCharacter(adir(i), names(i), 32)
   20 continue
C
      write(6,*) 'Directory of Mat-file:'
      do 30 i=1,ndir
        write(6,*) names(i)
   30 continue
C
      stat = matClose(mp)
      if (stat .ne. 0) then
        write(6,*) 'Error closing "matdemo.mat".'
        stop
      end if
C
C-----------------------------------------------------------------
C     Reopen file and read full arrays.
C-----------------------------------------------------------------
C
      mp = matOpen('matdemo.mat', 'r')
      if (mp .eq. 0) then
        write(6,*) 'Can"t open "matdemo.mat".'
        stop
      end if
C
C     Read directory.
C
      write(6,*) 'Getting full array contents:'
      pa = matGetNextMatrix(mp)
      do while (pa .ne. 0)
C
C     Copy name to character string.
C
        name = mxGetName(pa)
        write(6,*) 'Retrieved ', name
        write(6,*) '  With size ', mxGetM(pa), '-by-', mxGetN(pa)
        pa = matGetNextMatrix(mp)
```

```
      end do
C
      stat = matClose(mp)
      if (stat .ne. 0) then
        write(6,*) 'Error closing "matdemo.mat".'
        stop
      end if
      stop
C
      end
```

このプログラムをコンパイル、リンクした後で、結果を見ることができます。

```
matdemo2
Directory of Mat-file:
String
Numeric
Getting full array contents:
 1
Retrieved String
 With size   1-by-  33
 3
Retrieved Numeric
 With size   3-by-  3
```

MAT- ファイルプログラムのコンパイルとリンク

本節では、UNIX および Windows システムで MAT- ファイルプログラムをコンパイルし、リンクするために必要なステップを説明します。まず、浮動小数点の例外をマスクしないコンパイラに対して考慮すべき事柄に注目します。トピックスは以下の通りです。

- 浮動小数点の例外のマスク
- UNIX でのコンパイルとリンク
- Windows でのコンパイルとリンク

浮動小数点の例外のマスク

数学演算の中には、結果が不定値になるものがあります。たとえば、ゼロ割は IEEE の不定値 inf になります。浮動小数点の例外は、そのような演算が実行されるときに発生します。MATLAB は、inf や NaN のような不定値をサポートする IEEE モデルを使うので、MATLAB は浮動小数点の例外を利用不可能にしたり、マスクすることができます。

コンパイラの中には、デフォルトで浮動小数点の例外をマスクしないものがあります。これにより、そのようなコンパイラを使って構築された MAT- ファイルアプリケーションは、浮動小数点の例外が発生したときに停止します。その結果、MAT- ファイルアプリケーションが適切に実行されるためには、これらのコンパイラを使って浮動小数点の例外をマスクする際には予防策をとることが必要です。

注意　MATLAB ベースのアプリケーションは、浮動小数点の例外が生じません。浮動小数点の例外が発生した場合は、リンクしているサードパーティのライブラリが浮動小数点の例外ハンドリングが不可能でないかを確認してください。

浮動小数点の例外をマスクするプラットフォームおよびコンパイラは、以下の通りです。

Borland C++ Compiler on Windows

Windows プラットフォームで Borland C++ compiler を利用時に浮動小数点の例外をマスクするには、ユーザプログラムにコードを追加する必要があります。MATLAB API 関数の呼び出し前に、main() または WinMain() 関数の先頭につぎのコードを追加してください。

```
#include <float.h>
      .
      .
      .
_control87(MCW_EM,MCW_EM);
      .
      .
      .
```

UNIX でのコンパイルとリンク

ランタイムの UNIX では、API 共有ライブラリの場所をシステムに通知する必要があります。この節では、ユーザのシェルおよびシステムアーキテクチャに応じて必要な UNIX コマンドについて説明します。

ランタイムライブラリパスの設定

C シェルでは、ライブラリパスを設定するコマンドは、

```
setenv LD_LIBRARY_PATH <matlab>/extern/lib/$Arch:$LD_LIBRARY_PATH
```

Bourne シェルでは、ライブラリパスを設定するコマンドは、

```
LD_LIBRARY_PATH=<matlab>/extern/lib/$Arch:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

<matlab> は MATLAB ルートディレクトリで、$Arch はシステムアーキテクチャ (alpha, glnx86, sgi, sol2, hp700, ibm_rs) です。

環境変数 (この例題では LD_LIBRARY_PATH) は、プラットフォームにより異なります。つぎの表は、システムで利用する環境変数名の一覧です。

表 6-5: 環境変数名

アーキテクチャ	環境変数
HP700	SHLIB_PATH
IBM RS/6000	LIBPATH

C シェルに対しては ~/.cshrc、Bourne シェルに対しては ~/.profile のように、スタートアップスクリプトにコマンドを設定すると便利です。

オプションファイルの使用法

MATLAB は、MAT- ファイルアプリケーションを簡単にコンパイルおよびリンクするために、mex スクリプトを使うためのオプションファイル matopts.sh を提供しています。たとえば、例題の matcreat.c をコンパイル、リンクするには、つぎを使います。

```
mex -f <matlab>/bin/matopts.sh <pathname>/matcreat.c
```

ここで、<pathname> は、指定したファイルの完全なパスを指定します。

ユーザ固有のコンパイラまたはプラットフォームに対してオプションファイルを修正する必要がある場合は、-v スイッチを使ってカレントのコンパイラとリンカの設定を表示し、その後matopts.shファイルのローカルコピーに適切な変更を行ってください。

Windows でのコンパイルとリンク

Fortran または C MAT- ファイルプログラムをコンパイルおよびリンクするには、MAT オプションファイルと mex スクリプトを使用します。1-17 ページの表「オプションファイル」は、本リリースに含まれる MAT オプションファイルを示しています。これらのオプションファイルは、すべて <matlab>\bin\win32\mexopts にあります。

例題として、MSVC (Version 5.0) を使って Windows でスタンドアロン MAT アプリケーション matcreat.c をコンパイル、リンクするには、つぎのようにします。

```
mex -f <matlab>\bin\win32\mexopts\msvc50engmatopts.bat <pathname>\matcreat.c
```

ここで、<pathname> は、指定するファイルの完全なパスを指定します。ユーザ固有のコンパイラに対してオプションファイルを修正する必要がある場合は、-v スイッチを使ってカレントのコンパイラとリンカの設定を表示し、それからオプションファイルのローカルコピーで適切な変更を行ってください。

7

ActiveX と DDE のサポート

ActiveX は、オブジェクト指向のテクノロジで、ソフトウェア開発者が様々なベンダーのアプリケーション固有のコンポーネントを独自のアプリケーションのソリューションに統合することができるツールです、ActiveX Control コンポーネントは、MATLAB figure ウィンドウのような ActiveX コントロールコンテナに視覚的にまたプログラム的に統合することができます。ActiveX Automation を使って、MATLAB は他の ActiveX コンポーネントを制御したり、他の ActiveX コンポーネントが MATLAB を制御することができます。

Dynamic Data Exchange、または、DDE は、2 つのプログラムがお互いにデータを共有したり、直接コマンドを送信することを可能にする Windows の機能です。MATLAB は、このデータ交換を使って MATLAB と他の Windows アプリケーションとの間のアクセスを広範囲で可能にする機能を提供します。

本章では、つぎのトピックスについて説明します。

- MATLAB ActiveX の統合
- MATLAB ActiveX クライアントサポート
- イベントハンドラの作成
- ActiveX クライアントの情報
- MATLAB ActiveX Automation Server のサポート
- ActiveX Server の情報
- Dynamic Data Exchange (DDE)

MATLAB ActiveX の統合

ActiveX は、コンポーネント統合のための Microsoft Windows プロトコルです。ActiveX を使って、開発者およびエンドユーザは、様々なベンダーが作成したアプリケーション固有の ActiveX コンポーネントを選択し、アプリケーションのソリューションにシームレスに統合することができます。たとえば、1 つのアプリケーションがデータベースアクセス、数学的な分析、ビジネスグラフのプレゼンテーション機能が必要だとします。ActiveX を使って、開発者は、あるベンダによるデータベースアクセスコンポーネントを選択し、他のベンダーによるビジネスグラフコンポーネントを選択し、さらに他のベンダーが作成した数学分析パッケージにこれらを統合することができます。

ActiveX の概念と用語

COM

ActiveX は、*Component Object Model*、または、*COM* と呼ばれる共通の基礎をもつオブジェクト指向テクノロジの総称です。オブジェクト指向言語または環境は、オブジェクトの位置、事例、識別方法のような環境でのオブジェクトの特性を定義するためのオブジェクトモデルを持ちます。COM は、すべての ActiveX オブジェクトに対するオブジェクトモデルを定義します。

ActiveX インタフェース

ActiveX オブジェクトは、1 つまたは複数のインタフェースをサポートします。インタフェース は、メソッド、プロパティ、イベントの論理的に関連付けられた集合です。メソッドは、あるアクションを行うためのオブジェクトに対するリクエストである点で関数コールと似ています。プロパティは、テキストの色や、コントロールが動作しているファイル名のような、オブジェクトによって保持されている状態変数です。イベントは、クライアント (Handle Graphics® のコールバックと同じ) に送信する通知です。

たとえば、MATLAB に付属するサンプルのコントロールは、つぎのメソッド、プロパティ、イベントを持ちます。

- メソッド

  Redraw - コントロールを再描画します。
  Beep - コントロールをヒープします。
  AboutBox - コントロールの "About" ダイアログを表示します。

- プロパティ

  Radius (integer) - コントロールが描画される円の半径を設定します。
  Label (string) - コントロール内に描画されるテキスト

- イベント

  Click - ユーザがコントロール上をクリックすると実行

COM の 1 つの重要な特性は、複数のインタフェースをサポートするオブジェクトにおいてオブジェクトモデルを定義することです。インタフェースには、Microsoft によって定義されて、AcriveX の部分である標準 インタフェースや、個々のコンポーネントベンダによって定義されるカスタム インタフェースがあります。ActiveX オブジェクトを利用するために、サポートするカスタムインタフェース、インタフェースのメソッド、プロパティ、イベントについて学ぶ必要があります。ActiveX オブジェクトのベンダが、これらの情報を提供しています。

MATLAB ActiveX サポートの概要

MATLABは、ActiveX control containment と ActiveX Automationの2つのActiveXテクノロジをサポートします。ActiveX controls は、MATLAB figure ウィンドウのように、ActiveX control container に視覚的およびプログラム的に統合されるアプリケーションコンポーネントです。役に立つ ActiveX control の例は、Microsoft Internet Explorer Web Browser コントロール、シリアルポートアクセスに対する Microsoft Windows Communications コントロール、Visual Basic 開発環境で提供されるグラフィカルユーザインタフェースコントロールです。

ActiveX Automation を使って、MATLAB は他の ActiveX コンポーネントを制御したり、その逆を行うことができます。MATLAB が他のコンポーネントによって制御されているとき、オートメーション サーバ として機能しています。MATLAB が他のコンポーネントを制御しているときは、MATLAB はオートメーションクライアント で、それ以外のコンポーネントはオートメーション サーバ です。

MATLABオートメーションサーバ機能は、MATLABワークスペースでコマンドを実行し、直接ワークスペースとの行列のやりとりを行います。MATLAB オートメーションクライアント機能は、M- コードからオートメーションサーバをプログラム的に具体化し、操作することができます。ユーザは、オートメーションクライアント機能を使ってオートメーションサーバとしてコントロールを操作するので、MATLAB オートメーションクライアント機能は MATLAB control containment サポートのサブセットです。言い換えると、すべての ActiveX コントロールは、ActiveXオートメーションサーバであり、すべてのオートメーションサーバが必ずしもコントロールであるわけではありません。

一般に、コントロールでないサーバは、物理的または視覚的にクライアントアプリケーションに組み込まれません (MATLAB は良い例です。-- MATLAB 自身はコントロールではありませんが、サーバです。そのため、MATLAB は物理的に他のクライアントに組み込めません。しかし、MATLAB は *container* なので、他の ActiveX コントロールは、MATLAB に組み込まれます)。

さらに、MATLAB は、スクリーン上に円を描画し、テキストを表示する簡単な ActiveX コントロールの例題を提供しています。MATLAB ユーザはこの例題を使って、既知のコントロールを使って MATLAB の ActiveX コントロールサポートを試してみることができます。詳細は、「MATLAB コントロールの例」を参照してください。

MATLAB ActiveX クライアントのサポート

MATLABまたは他のActiveXクライアントでActiveXコンポーネントを使うためには、最初にそのオブジェクトのドキュメントを参考にし、オブジェクトが利用するインタフェース名、メソッド名、プロパティ名、イベント名、オブジェクト名 (*ProgID* として知られています) を見つける必要があります。この情報を得ると、ActiveX クライアントサポートを使って MATLAB とオブジェクトを統合できます。

ActiveX オブジェクトの使用法

MATLAB ActiveX クラスを作成することにより、MATLAB において ActiveX コントロールまたはサーバを作成します。各インスタンスは、オブジェクトについて 1 つのインタフェースを表わします。

注意　このマニュアルでは、一般の ActiveX コントロール / サーバを参照するのに ActiveX を用い、MATLAB クラス / オブジェクトを参照するのに ActiveX を用います。

ActiveX オブジェクトの作成

ActiveX オブジェクトを作成するために、始めに 2 つのコマンドが利用されます。

actxcontrol	ActiveX コントロールを作成します。
actxserver	ActiveX オートメーションサーバを作成します。

インタフェースの操作

インタフェースを表わす ActiveX オブジェクトを作成すると、オブジェクトにおいてメソッドを呼び込んで操作することによって、様々なアクションを実行できます。つぎのメソッドは、ActiveX クラスに対して実現されます。

メソッド	説明
delete	ActiveX オブジェクトを削除します。

get	インタフェースからプロパティ値を取得、またはプロパティのリストを表示します。
invoke	インタフェースにおいてメソッドを呼び込む、またはメソッドのリストを表示します。
load	ファイルから ActiveX オブジェクトを初期化します。
move	親ウィンドウの ActiveX コントロールを移動またはリサイズします。
propedit	組み込みプロパティページを表示するかどうかをコントロールに指示します。
release	ActiveX オブジェクトを開放します。
save	ActiveX コントロールオブジェクトをファイルに保存します。
send	イベントのリストを表示します。
set	インタフェースのプロパティを設定します。

作成コマンド actxcontrol と actxserver は、作成されたオブジェクトに対するデフォルトのインタフェースを表わす MATLAB ActiveX オブジェクトを出力します。しかし、これらのオブジェクトは、他のインタフェースを持つ場合があります。インタフェースは、既存のインタフェースにおいてメソッドを呼び込んだり、既存のインタフェースからプロパティを取得することによって得ることが可能 (および一般的) です。ActiveX の get および invoke メソッドは、新たな ActiveX オブジェクトを自動的に作成、出力して、これらのインタフェースを表わします。

インタフェースが必要でないときには、release メソッドを使ってインタフェースを開放します。コントロール全体またはサーバが必要でないときには、delete コマンドを使って削除します。詳細は、「インタフェースの開放」を参照してください。

ActiveX クライアントリファレンス

本節は、ActiveX オブジェクトを作成し、インタフェースを操作するコマンドのリファレンスページです。

目的

figure ウィンドウで ActiveX コントロールを作成します。

表示

```
h =actxcontrol (progid [, position [, handle ...
    [,callback |{event1 eventhandler1; ...
    event2 eventhandler2; …}]]])
```

引数

progid
作成するコントロール名を示す文字列。コントロールのベンダがこの文字列を提供します。

position
コントロールの x および y の位置と xsize と ysize を含む位置ベクトル、ピクセル単位で [x y xsize ysize] と表わされます。デフォルトは、[20 20 60 60] です。

handle
コントロールが作成される figure ウィンドウの Handle Graphics ハンドル。コントロールが非可視である場合は、非可視の figure ウィンドウのハンドルを使います。デフォルトは gcf です。

callback
可変数の引数を受け取る M- ファンクション名。この関数は、コントロールがイベントをトリガするときにコールされます。引数は、MATLAB 文字列に変換されます。第一引数は、常にトリガされたイベントの数値を表わす文字列です。これらの数値は、コントロールによって定義されます (コントロールイベントのハンドリングの詳細は、「イベントハンドラを書く」" を参照してください)。

event
数値または名前で指定されたトリガされたイベント。

eventhandler
可変数の引数を受け取る M- ファンクション名。この関数は、コントロールが関連するイベントをトリガするときにコールされます。第一引数はActiveXオブジェクト、第二引数はトリガされたイベントの数値を表わします。第二引数は、" コールバック "M- ファイルスタイルのように文字列には変換されません。これらの値は、コントロールによって定義されます。コントロールイベントのハンドリングに関する詳細は、「イベントハンドラを書く」を参照してください。

注意　イベントのハンドリング方法には 2 種類あります。単一のコールバックを作成するか、イベントとイベントハンドラの組を含むセル配列を指定することができます。セル配列形式では、数値または名前でイベントを指定します。コントロールは、イベント数と名前を定義します。セル配列で指定できる組の数に制限はありません。単一のコールバックメソッドを使うほうが簡単な場合がありますが、セル配列法を使うことによって、より良いパフォーマンスとなる効率的なコードを作成します。

出力

コントロールまたはサーバに対するデフォルトのインタフェースを表わす MATLAB ActiveX オブジェクト。このオブジェクトについて get, set, invoke, propedit, release, delete メソッドを使います。このコールが失敗した場合は、MATLAB エラーが生成されます。

詳細

figure ウィンドウの特定の位置に ActiveX コントロールを作成します。親の figure ウィンドウが非可視である場合は、コントロールは非可視になります。出力される MATLAB ActiveX オブジェクトは、コントロールに対するデフォルトインタフェースを表わします。このインタフェースが必要でないときには、インタフェースが利用するメモリとリソースを開放するために、release をコールして開放しなければなりません。インタフェースの開放は、コントロール自身は削除しません (delete コマンドを使ってコントロールを削除してください)。

コールバックのイベントハンドラの例は、toolbox\matlab\winfun の sampev.m を参照してください。

例題

コールバックスタイル：

```
f = figure ('pos', [100 200 200 200]);
%  figure を描画するためにコントロールを作成
h = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], ...
        gcf, 'sampev')
```

セル配列スタイル

```
h = actxcontrol ('SELECTOR.SelectorCtrl.l', ...
[0 0 200 200], f, {-600 'myclick'; -601 'my2click'; ...
-605 'mymoused'})

h = actxcontrol ('SELECTOR.SelectorCtrl.l', ...
[0 0 200 200], f, {'Click' 'myclick'; ...
```

```
'DblClick' 'my2click'; 'MouseDown' 'mymoused'})
```

ここで、イベントハンドラ myclick.m, my2click.m, mymoused.m は、

```
function myclick(varargin)
disp('Single click function')

function my2click(varargin)
disp('Double click function')

function mymoused(varargin)
disp('You have reached the mouse down function')
disp('The X position is: ')
varargin(5)
disp('The Y position is: ')
varargin(6)
```

セル配列の組を使って、モニターしたいすべてのイベントに対して同じイベントハンドラを使うことができます。応答時間は、コールバックスタイルを使うよりも良くなります。

たとえば、

```
h = actxcontrol('SELECTOR.SelectorCtrl.1', ...
[0 0 200 200], f, {'Click' 'allevents'; ...
'DblClick' 'allevents'; 'MouseDown' 'allevents'})
```

allevents.m は、

```
function allevents(varargin)
if (varargin{2} = -600)
    disp ('Single Click Event Fired')
elseif (varargin{2} = -601)
    disp ('Double Click Event Fired')
elseif (varargin{2} = -605)
    disp ('Mousedown Event Fired')
end
```

目的

ActiveX オートメーションサーバを作成し、サーバのデフォルトインタフェースに対して ActiveX オブジェクトを出力します。

表示

h = actxserver (progid [, *MachineName*])

引数

progid
これは、具体化するコントロール名の文字列です。この文字列は、コントロールまたはサーバのベンダによって提供され、ベンダのドキュメントから得ることができます。たとえば、Microsoft Excel に対する progid は Excel.Application です。

MachineName
これは、サーバが起動されるリモートマシン名です。この引数はオプションで、Distributed Component Object Model (DCOM) をサポートする環境でのみ用いられます (下記参照)。これは、IP アドレスまたは DNS 名です。

出力

サーバのデフォルトインタフェースを表わすActiveXオブジェクト。このオブジェクトについて、get, set, invoke, release, delete メソッドを使います。このコールが失敗した場合は、MATLAB エラーが生成されます。

詳細

ActiveX オートメーションサーバを作成し、サーバのデフォルトインタフェースを表わす MATLAB ActiveX オブジェクトを出力します。ローカル / リモートサーバは、異なるアドレス空間 (かつ、異なるマシン上で) 起動され、MATLAB プロセスの一部でない点がコントロールと異なります。さらに、表示するユーザインタフェースは、別のウィンドウにあり、MATLAB プロセスに接続されません。ローカルサーバの例は、Microsoft Excel や Microsoft Word です。オートメーションサーバは、コールバックやイベントハンドラを使わないことに注意してください。

例題

```
% Microsoft Excel を立ち上げ、メインフレームウィンドウを見えるようにする
h = actxserver ('Excel.Application')
set (h, 'Visible', 1);
```

delete

目的

ActiveX コントロール、または、サーバを削除します。

表示

delete (a)

引数

a
actxcontrol, actxserver, get, invoke から出力された ActiveX オブジェクト。

詳細

ActiveX コントロール、または、サーバを削除します。これは、インタフェースのみを開放して無効にするインタフェースの開放とは異なります。delete は、すべてのインタフェースを開放し、ActiveX サーバまたはコントロールを削除します。

例題

```
delete (a)
```

目的

インタフェースからプロパティ値を取得するか、プロパティのリストを取得します。

表示

v = get (a [, 'propertyname' [, arg1, arg2, …]])

引数

a
actxcontrol, actxserver, get, invoke から出力された ActiveX オブジェクト。

propertyname
取得するプロパティ値の名前の文字列。

arg1, …, argn
取得されるプロパティが要求する引数。プロパティは、引数を持つことができる点がメソッドと同じです。

出力

プロパティの値またはプロパティのリスト (get(a) 形式を利用した場合)。この値の意味とタイプは、取得される個々のプロパティに依存します。オブジェクトのドキュメントには、出力値の意味が記述されています。ActiveX データタイプの変換方法の説明は、「データ変換」を参照してください。

詳細

インタフェースからプロパティ値を取得します。

注意 actxcontrol の出力値を関数 get の引数として使って、呼び出されるコントロールとメソッドのプロパティのリストを取得できます。

例題

1 つのプロパティ値を取得します。

```
% get the string value of the 'Label' property
s = get (a, 'Label');
```

プロパティのリストを取得します。

```
get(a)

AutoAlign = [-1]
AutoAngle = [-1]
AutoAngleConfine = [0]
AutoOffset = [-1]
.
```

```
.
.
SelectionOffsetY = [0]
SelectionRadius = [0.8]
Selections = [4]
Value = [0]
```

目的

オブジェクトのインタフェースにおいてメソッドを呼び出し、メソッドの値があれば取得するか、またはメソッドのリストを表示します。

表示

```
v = invoke (a [, 'methodname' [, arg1, arg2, …]])
```

引数

a
actxcontrol, actxserver, get, invoke から出力された ActiveX オブジェクト。

methodname
呼び出されるメソッド名の文字列。

arg1, …, argn
呼び出されるメソッドが要求する引数。

出力

メソッドが出力する値またはメソッドのリスト（invoke(a) の形式を利用する場合）。値のデータタイプは、呼び出される固有のメソッドに依存し、コントロールまたはサーバによって決定されます。メソッドがインタフェースを出力する場合（ActiveX のドキュメントでは、interface または Idispatch * と記述されています）、このメソッドは、出力されたインタフェースを表わす新たな MATLAB ActiveX オブジェクトを出力します。ActiveX データタイプの変換方法の説明は、「データ変換」を参照してください。

詳細

オブジェクトのインタフェースにおいてメソッドを呼び出し、メソッドの出力値があれば取得します（メソッドの中には出力値を持たないものがあります）。

例題

メソッドを呼び出します。

```
f = figure ('pos', [100 200 200 200]);
% figure を描画するためのコントロールを作成
h = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], f)
set (h, 'Radius', 100);
v = invoke (h, 'Redraw')
```

メソッドのリストを表示します。

```
invoke(h)

AboutBox = Void AboutBox ()
ShowPropertyPage = Void ShowPropertyPage ()
```

load

目的

ファイルから ActiveX オブジェクトを初期化します。

表示

load(h, filename)

引数

h
MATLAB ActiveX オブジェクト。

filename
データの絶対パス名。

出力

void

詳細

load は、MATLAB ActiveX object H によって表わされるインタフェースに関連する ActiveX オブジェクトをファイルから初期化します。ファイルは、同じコントロールのインスタンスをシリアライズすることによってあらかじめ作成されていなければなりません。

例題

```
h = actxcontrol('MwSamp.mwsampctrl.1');
    load(h, 'c:\temp\mycontrol.acx');
```

目的

ActiveX コントロールを親ウィンドウで移動またはリサイズします。

表示

```
move(h, pos)
```

引数

h
MATLAB ActiveX コントロールオブジェクト。

pos
位置。

出力

現在の位置。

詳細

move(h, position) は、コントロールを新規の位置に移動します。
position = move(h) は、現在の位置を出力します。

例題

この例題は、コントロールを移動します。

```
h = actxcontrol('MWSamp.mwsampctrl.1');
    move(h, [100 100 200 200]);
    pos = move(h);  % pos は、[100 100 200 200] になります。
```

この例題は、常に図全体になるようにコントロールをリサイズします。MATLAB または M- ファイル内で以下を実行してください。

```
f = figure('Position', [100 100 200 200]);
h = actxcontrol('MWSAMP.MwsampCtrl.1', [0 0 200 200]);
set(f, 'ResizeFcn', 'resizectrl')
```

以下を含むスクリプト resizectrl.m を作成します。

```
% figure ウィンドウの新しい位置とサイズを取得
 fpos = get(gcbo, 'position');

% コントロールをリサイズ
 move(h, [0 0 fpos(3) fpos(4)]);
```

propedit

目的

コントロールを要求して組み込みプロパティページを表示します。

表示

propedit (a)

引数

a
actxcontrol, get, invoke が出力したインタフェースのハンドル。

詳細

コントロールを要求して組み込みのプロパティページを表示します。コントロールの中には、組み込みプロパティページを持たないものがあります。これらのオブジェクトに対しては、このコマンドは失敗します。

例題

```
propedit (a)
```

目的

インタフェースを開放します。

表示

release (a)

引数

a
開放するインタフェースを表わす ActiveX オブジェクト。

詳細

インタフェースとインタフェースが用いるすべてのリソースを開放します。各インタフェースのハンドルは、プロパティの操作とメソッドの呼び出しが終了したときに開放する必要があります。インタフェースが開放されると、有効でなくなり、インタフェースを表わすその後の MATLAB での ActiveX の操作は、エラーになります。

注意　インタフェースを開放しても、そのオブジェクトについて他のインタフェースがアクティブである場合があるので、コントロールを削除しません (delete を参照)。詳細は、「インタフェースの開放」を参照してください。

例題

```
release (a)
```

save

目的

ActiveX コントロールオブジェクトをファイルに保存します。

表示

save(h, filename)

引数

h
MATLAB ActiveX オブジェクト。

filename
シリアライズされたデータの絶対パス名。

詳細

MATLAB ActiveX object H で表わされるインタフェースに関連する ActiveX オブジェクトをファイルに保存します。filename は、シリアライズされたデータの絶対パス名です。

例題

```
h = actxcontrol('MwSamp.mwsampctrl.1');
    save(h, 'c:\temp\mycontrol.acx');
```

目的

コントロールがトリガするイベントのリストを出力します。

表示

send (a)

引数

a
actxcontrol が出力した ActiveX オブジェクト。

詳細

コントロールが send するイベントのリストを表示します。

例題

```
send (a)

Change = Void Change ()
Click = Void Click ()
DblClick = Void DblClick ()
KeyDown = Void KeyDown (Variant(Pointer), Short)
KeyPress = Void KeyPress (Variant(Pointer), Short)
KeyUp = Void KeyUp (Variant(Pointer), Short)
MouseDown = Void MouseDown (Short, Short, Vendor-Defined,
    Vendor-Defined)
MouseMove = Void MouseMove (Short, Short, Vendor-Defined,
    Vendor-Defined)
MouseUp = Void MouseUp (Short, Short, Vendor-Defined,
    Vendor-Defined)
```

set

目的

インタフェースのプロパティを特定の値に設定します。

表示

set (a [, 'propertyname' [, value [, arg1, arg2, …]]])

引数

a
actxcontrol, actxserver, get, invoke によって出力された ActiveX オブジェクトのハンドル。

propertyname
設定するプロパティ名の文字列。

value
インタフェースのプロパティが設定される値。

arg1, …, argn
プロパティが要求する引数。プロパティは、引数を持つことができる点がメソッドと同じです。

出力

set の出力値はありません。

詳細

インタフェースのプロパティを特定の値に設定します。ワークスペース行列の ActiveX データタイプへの変換方法に関する情報は、「データ変換」を参照してください。

例題

```
f = figure ('pos', [100 200 200 200]);
%figure を描画するためにコントロールを作成
a = actxcontrol ('MWSAMP.MwsampCtrl.1', [0 0 200 200], f)
set (a, 'Label', 'Click to fire event');
set (a, 'Radius', 40);
invoke (a, 'Redraw');
```

イベントハンドラを書く

ActiveXイベントは、なにか興味のあることが発生したことをcomtainerに通知したいときに呼び出されます。たとえば、多くのコントロールは、ユーザがコントロール上でシングルクリックしたときにイベントをトリガします。MATLAB では、コントロールが作成されるとき、actxcontrol コマンドの最後の引数として、またはイベントハンドラを含むセル配列としてオプションでコールバック（*イベントハンドラ関数* としても知られています）を与えます。

```
h = actxcontrol (progid [, position [, handle ...
[, callback | {event1 eventhandler1; event2 eventhandler2; …}]]])
```

イベントハンドラ関数（callback）は、コントロールがイベントをトリガするときにコールされます。イベントハンドラ関数は、つぎの形式で、可変数の引数を受け取る M- ファンクションでなければなりません。

```
function event (varargin)
if (str2num(vararg{1}) == -600)
   disp ('Click Event Fired');
end
```

この関数に渡されるすべての引数は、MATLAB 文字列です。イベントハンドラの第一引数は、イベントハンドラをコールさせるイベント数を表わす文字列です。残りの引数は、イベントと共にコントロールによって渡される値です。これらの値は、用いられるイベントとコントロールにより異なります。呼び出しをコントロールするイベントのリストと対応するイベント数とパラメータは、コントロールのドキュメントから得る必要があります。MATLAB でイベントを利用するには、イベントハンドラでそれらを使うために、コントロールがイベントに対して使う数値を見つける必要があります。

セル配列形式で使われるイベントハンドラは、コールバック形式で使われるものとわずかに異なります。セル配列形式の第一引数は、オブジェクト自身を参照します。第二引数はイベントの数で、コールバック形式と異なり、数値を表わす文字列ではありません（その後の引数は、イベントにより異なります）。

つぎのイベントハンドラの例 myclick.m, my2click.m, mymoused.m は、actxcontrol の例題に対応します。

```
function myclick(varargin)
disp('Single click function')

function my2click(varargin)
disp('Double click function')

function mymoused(varargin)
disp('You have reached the mouse down function')
disp('The X position is: ')
varargin(5)
disp('The Y position is: ')
varargin(6)
```

セル配列の組を使って、モニターしたいすべてのイベントに対して同じイベントハンドラを使うことができます。応答時間は、コールバック形式を使うよりも良くなります。

たとえば、

```
h = actxcontrol('SELECTOR.SelectorCtrl.1', [0 0 200 200], f, ...
{'Click' 'allevents'; 'DblClick' 'allevents'; ...
'MouseDown' 'allevents'})
```

allevents.m は、

```
function allevents(varargin)
if (varargin{2} = -600)
    disp ('Single Click Event Fired')
elseif (varargin{2} = -601)
    disp ('Double Click Event Fired')
elseif (varargin{2} = -605)
    disp ('Mousedown Event Fired')
end
```

注意　　MATLAB は、リファレンスまたはイベントからの出力によって渡されるイベント引数をサポートしません。

ActiveX クライアントの情報

インタフェースの開放

各 ActiveX オブジェクトは、1 つまたは複数のインタフェースをサポートします。MATLAB では、インタフェースは ActiveX クラスのインスタンスによって表わされます。MATLAB ワークスペースに有効なインタフェースオブジェクトを取得する方法は、3 種類あります。

- actxcontrol/actxserver からの出力値
- get によるプロパティの出力値
- invoke によるメソッド呼び出しからの出力値

いずれの場合でも、インタフェースがワークスペース内の acriveX オブジェクトによって表わされると、終了時に開放する必要があります。インタフェースのハンドルの開放に失敗すると、メモリやリソースを消費します。その代わりに、delete コマンドを有効なインタフェースオブジェクトに対して使うと、オブジェクトに対する全てのインタフェースは自動的に開放され (そのため無効になります)、ActiveX サーバまたはコントロールは削除されます。

MATLAB は、コントロールを含む figure ウィンドウが削除またはクローズされたときに、自動的に ActiveX に対するすべてのインタフェースを開放します。MATLAB は、MATLAB がシャットダウンされるときに、ActiveX オートメーションサーバに対するすべてのハンドルを自動的に開放します。

ActiveX Collections を使って

ActiveX collections は、繰り返しが可能な ActiveX オブジェクトの関連するグループをサポートする方法です。collection は、collection の中の item 数を含む Count プロパティ (参照のみ) と、collection から単一のアイテムを取得することが可能な Item メソッドをもつ特殊なインタフェースです。

Item メソッドは、インデックス付けられています。これは、collection のどのアイテムが要求されるかを指定する引数が必要であることを意味します。インデックスのデータタイプは、特定の collection に対して適切な任意のデータタイプで、collection をサポートするコントロールまたはサーバに固有です。整数のインデックスが一般的ですが、文字列の値にすることが可能です。Item メソッドの返り値は、インタフェースである場合があります。すべてのインタフェースと同様に、このインタフェースは終了時に開放されます。

つぎの例は、collection のメンバを繰り返します。collection の各メンバは、インタフェースです (Plot と呼ばれ、hPlot という MATLAB の ActiveX オブジェクトに

よって表わされます)。特に、この例は Plot インタフェースの collection を繰り返し、各インタフェースについて Redraw メソッドを呼び出し、各インタフェースを開放します。

```
hCollection = get (hControl, 'Plots');
for i=1:get (hCollection, 'Count')
hPlot  = invoke (hCollection, 'Item', i);
      invoke (hPlot, 'Redraw');
      release (hPlot);
end;
release (hCollection);
```

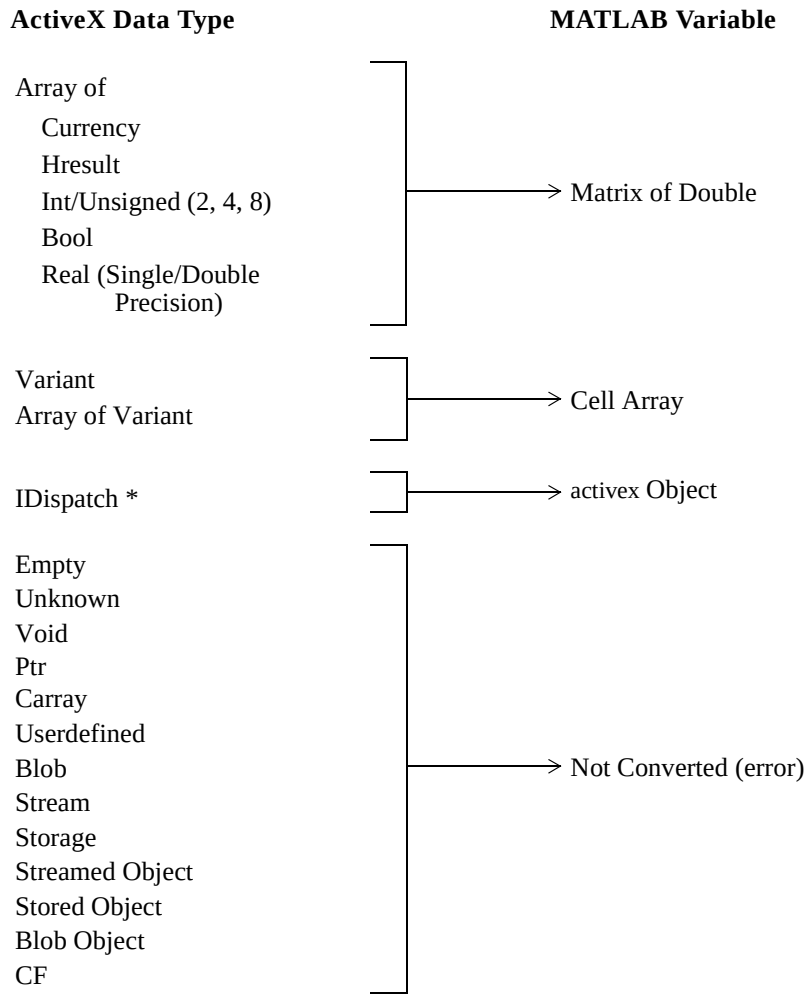
データ変換

ActiveX は、多くの異なるデータ書式やタイプを定義するので、ActiveX オブジェクトから MATLAB ワークスペース変数への変換方法を知っている必要があります。ActiveX オブジェクトからのデータは、以下の場合に変換する必要があります。

- プロパティ値が取得されるとき
- 値がメソッド呼び出しから出力されるとき

つぎの表は、ActiveX データタイプの MATLAB ワークスペース変数への変換方法を示しています。

ActiveX Data Type	**MATLAB Variable**
String File Time Error Decimal Date	→ MATLAB String
Currency Hresult Int/Unsigned (2, 4, 8) Bool Real (Single/Double Precision)	→ Scalar Double
Null	→ NaN



MATLAB を DCOM Server Client として使う

Distributed Component Object Model(DCOM) は、ActiveX クライアントがネットワーク上でリモート ActiveX オブジェクトを利用するためのオブジェクト分散メカニズムです。このマニュアルの記述時では、DCOM は NT 4.0 に付属し、Microsoft for Windows 95 から取得することができます。

MATLAB は、Windows NT のみ DCOM サーバとしてテストしています。さらに、MATLAB を起動しているオペレーティングシステムが DCOM が利用可能である場合は、リモートオートメーションサーバを持つ DCOM クライアントとして使うことができます。

注意　MATLAB をリモート DCOM サーバとして使う場合は、全ての MATLAB ウィンドウはリモートマシン上に現れます。

MATLAB ActiveX サポートの制限

MATLAB ActiveX サポートの制限のリストを示します。

- MATLAB は、インデックス付きの collection のみをサポートします。
- ActiveX コントロールは、figure ウィンドウと共に印刷されません。
- MATLAB　は、リファレンスによって渡されたイベント引数をサポートしません。
- MATLAB は、イベントハンドラ関数からの返り値をサポートしません。
- コントロールの位置ベクトルは、変更または確認できません。
- MATLAB は、リファレンスによって渡されるメソッドまたはイベント引数をサポートしません。

MATLAB コントロールの例

MATLAB には、スクリーン上に円を描画し、テキストを表示し、コントロール上でクリックしたときにクリックされた イベントを実行する簡単な ActiveX の例が付属しています。このコントロールを使って、既知のコントロールを使った ActiveX コントロールのサポートを簡単に試みることができます。コントロールは、winfun ディレクトリの mwsamp ファイルを実行して作成できます。

コントロールは、コントロールのタイプライブラリ と共に MATLAB の bin または実行ファイルディレクトリに格納されます。*タイプライブラリ*は、コントロールの機能を解読するために ActiveX ツールが用いるバイナリファイルです。

MATLAB をオートメーションクライアントとして利用

この例は、MATLAB をオートメーションクライアントとして利用し、Microsoft Excel をサーバとして利用します。この例題は、典型的な関数の概要を提供しま

す。さらに、他のアプリケーションのオートメーションインタフェースの利用の良い例です。

```
% MATLAB ActiveX オートメションクライアントの例題
%
% Excel を開き、ワークブックを追加し、アクティブなワークシートを変更
% 配列を取得、配置し、保存

% 最初に、Excel Server を開きます。
Excel = actxserver('Excel.Application');
set(Excel, 'Visible', 1);

% 新しいワークブックを挿入
Workbooks = Excel.Workbooks;
Workbook = invoke(Workbooks, 'Add');

% 二番目のシートをアクティブにします。
Sheets = Excel.ActiveWorkBook.Sheets;
sheet2 = get(Sheets, 'Item', 2);
invoke(sheet2, 'Activate');

% アクティブなシートにハンドルを取得
Activesheet = Excel.Activesheet;

% MATLAB 配列を Excel に配置
A = [1 2; 3 4];
ActivesheetRange = get(Activesheet,'Range','A1','B2');
set(ActivesheetRange, 'Value', A);

% 範囲を再取得。セル範囲は違うデータタイプを
% 含むことができるので、範囲は、セル配列になります。
Range = get(Activesheet, 'Range', 'A1', 'B2');
B = Range.value;

% double 行列に変換し、セル配列はスカラのみを含む
%
B = reshape([B{:}], size(B));

% ワークブックを保存
invoke(Workbook, 'SaveAs', 'myfile.xls');
```

```
% ワークブックを保存せずに、プロンプトを表示しないためには、
% 以下のコードを取り除きます。
% Workbook.Saved = 1;
% invoke(Workbook, 'Close');

% Excel 終了
invoke(Excel, 'Quit');
```

MATLAB ActiveX Automation Server のサポート

Microsoft Windows版のMATLABは、ActiveX Automation server機能をサポートします。Automation は、アプリケーションまたはコンポーネント (コントローラ) が他のアプリケーションまたはコンポーネント (サーバ) を制御することができる ActiveX プロトコルです。そのため、MATLAB は、Automation Controller になりうる任意の Windows プログラムによって起動され、制御されることが可能です。Automation Controller になりうるアプリケーションの例は、Microsoft Excel, Microsoft Access, Microsoft Projectおよび多くのVisual BasicおよびVisual C++プログラムです。Automation を使って、MATLAB コマンドを実行し、MATLAB ワークスペースと mxArray とのやり取りが可能です。

MATLAB をオートメーションサーバとして使うためには、つぎのステップ 1 および 2 を使います。

1 ActiveX Automation server の呼び出し

コントローラのドキュメントの ActiveX Automation サーバの呼び出し方法を参照します。レジストリ内にある MATLAB ActiveX オブジェクト名は、Matlab .Application です。MATLAB サーバを呼び出す方法は、選択するコントローラにより異なりますが、すべてのコントローラでサーバを識別するためにこの名前が必要です。

2 コマンド文字列で Execute メソッドを利用

MATLAB の ActiveX Automation インタフェースは、複数のメソッドをサポートします。以下は、MATLAB Automation Execute メソッドを呼び出し、Microsoft Excel または任意の Visual Basic または Visual Basic for Applications (VBA) イネーブルなアプリケーションにおいて機能する Visual Basic のコードの一部です。Execute メソッドは、コマンド文字列を引数として受け取り、結果を文字列として出力します。コマンド文字列は、通常、コマンドウィンドウにタイプされる任意のコマンドでかまいません。結果には、コマンドウィンドウに文字列を実行した結果として表示された、エラーを含む出力が含まれます。

```
Dim MatLab As Object

Dim Result As String
Set MatLab = CreateObject("Matlab.Application")
Result = MatLab.Execute("surf(peaks)")
```

注意　上記の最初のステートメントは、アプリケーション全体の範囲を保持するために、一般の宣言部分で宣言されます。

MATLAB ActiveX Automation メソッド

本節では、MATLAB Automation Server がサポートするメソッドについて示します。引数と返り値に対するデータタイプは、ActiveX Automation データタイプとして表わされ、これは ActiveX Automation プロトコルで定義される言語と独立のタイプです。たとえば、BSTR は Automation タイプとして定義されたワイドキャラクタ文字列のタイプで、文字列を保存するために Visual Basic で使うデータ書式と同じです。これらの宣言と操作法の詳細はコントローラ固有ですが、ActiveX 準拠コントローラはこれらのデータタイプをサポートします。

BSTR Execute([in] BSTR Command);

このコマンドは、MATLAB コマンドウィンドウプロンプトでタイプされるコマンドを含む単一の文字列 (Command) を持ちます。MATLAB は、コマンドを実行し、結果を文字列として出力します。コマンドにより生成された figure ウィンドウは、コマンドがコマンドウィンドウまたは M- ファイルから直接実行されたかのようにスクリーンに表示されます。Visual Basic の例を以下に示します。

```
Dim MatLab As Object

Dim Result As String
Set MatLab = CreateObject("Matlab.Application")
Result = MatLab.Execute("surf(peaks)")
```

```
void GetFullMatrix(
                [in] BSTR Name,
                [in] BSTR Workspace,
                [in, out] SAFEARRAY(double)* pr,
                [in, out] SAFEARRAY(double)* pi);
```

注意　上記の最初のステートメントは、アプリケーション全体の範囲を保持するために一般の宣言部分で宣言されます。

このメソッドは、ワークスペースから 1 次元または 2 次元の実数または虚数配列 mxArray を読み込みます。実部と (オプションの) 虚部は、double の別々の配列に読み込まれます。

Name

読み込まれる mxArray 名を識別します。

Workspace

mxArray を含むワークスペースを識別します。デフォルトの MATLAB ワークスペースから mxArray を読み込むためには、ワークスペース名 "base" を使ってください。mxArray をグローバルの MATLAB ワークスペースに置くためには、ワークスペース名 "global" を使ってください。" 呼び出し側の " ワークスペースは、MEX- ファイル以外で使われる場合は API では意味を持ちません。

pr

読み込まれる mxArray と同じサイズの実数配列。この配列は、mxArray の実数値を含みます。

pi

読み込まれる mxArray と同じサイズの実数配列。この配列は、mxArray の虚数値を含みます。要求される mxArray が複素数でなければ、空配列が渡されなければなりません。Visual Basic では、空配列は、Dim Mempty() as Double と宣言されます。Visual Basic の例を以下に示します。

```
Dim MatLab As Object

Dim Result As String
Dim MReal(1, 3) As Double
Dim MImag() As Double
Dim RealValue As Double
Dim i, j As Integer

rem We assume that the connection to MATLAB exists.
Result = MatLab.Execute("a = [1 2 3 4; 5 6 7 8;]")
Call MatLab.GetFullMatrix("a", "base", MReal, MImag)

For i = 0 To 1
    For j = 0 To 3
    RealValue = MReal(i, j)
    Next j
Next i
```

```
void PutFullMatrix(
                [in] BSTR Name,
                [in] BSTR Workspace,
                [in] SAFEARRAY(double) pr,
                [in] SAFEARRAY(double) pi);
```

注意　上記の最初のステートメントは、アプリケーション全体の範囲を保持するために一般の宣言部分で宣言されます。

このメソッドは、ワークスペースに 1 次元または 2 次元の実数または虚数配列 mxArray を置きます。実部と (オプションの) 虚部は、double の別々の配列に渡されます。

Name

読み込まれる mxArray 名を識別します。

Workspace

mxArray が置かれるワークスペースを識別します。デフォルトの MATLAB ワークスペースに mxArray を置くためには、ワークスペース名 "base" を使ってください。グローバルの MATLAB ワークスペースに mxArray を置くためには、ワークスペース名 "global" を使ってください。" 呼び出し側の " ワークスペースは、MEX-ファイル以外で使われる場合は、API では意味を持ちません。

pr

mxArray の実数値を含む実数配列。

pi

mxArray の虚数値を含む実数配列。送信される mxArray が複素数でない場合は、このパラメータに対して空配列が渡されなければなりません。Visual Basic では、空配列は Dim Mempty() as Double と宣言されます。このメソッドの Visual Basic の例を以下に示します。

```
Dim MatLab As Object

Dim MReal(1, 3) As Double
Dim MImag() As Double
Dim i, j As Integer
For i = 0 To 1
    For j = 0 To 3
```

```
    MReal(i, j)  = I * j;
    Next j
Next I
rem We assume that the connection to MATLAB exists.
Call MatLab.PutFullMatrix("a", "base", MReal, MImag)
```

注意　　上記の最初のステートメントは、アプリケーション全体の範囲を保持するために一般の宣言部分で宣言されます。

ActiveX Server の情報

MATLAB ActiveX Server の起動

MATLAB がオートメーションサーバとして動作するためには、コマンドライン引数 /Automation を使って起動されなければなりません。ActiveX の接続がコントローラにより行われるとき、Microsoft Windows は自動的に起動します。しかし、MATLAB が既に実行されており、このパラメータなしで起動されていたとすれば、オートメーションコントローラによるサーバとしての MATLAB への接続要求により、Windows は /Automation パラメータを使って別の MATLAB を起動します。これは、実行中の対話的な MATLAB セッションにコントローラが干渉されることを防ぎます。

共有または専用サーバの指定

MATLAB Automation サーバは、共有または専用の 2 つのモードのいずれかで起動できます。専用サーバは単一のクライアント専用であり、共有サーバは複数のクライアントにより共有されます。

モードは、MATLAB を起動するクライアントが用いる Program ID (ProgID) により決定されます。ProgID, Matlab.Application(またはバージョン固有の ProgID, Matlab.Application.5) は、デフォルトのモード (共有) を指定します。専用サーバを指定するには、ProgID, Matlab.Application.Single, (またはバージョン固有の ProgID, Matlab.Application.Single.5) を使います。用語 Single は、サーバに接続可能なクライアント数を参照します。

MATLAB が共有サーバとして起動されると、共有サーバの ProgID を使って MATLAB への接続を要求するすべてのクライアントは、既に起動されている MATLAB に接続されます。言い換えると、共有サーバを指定する ProgID を使うすべてのクライアントが共有しているので、起動できる共有サーバは1つだけです。

注意　クライアントは、対話的な MATLAB には接続しません。つまり、マニュアルで /Automation コマンドラインフラグなして起動された MATLAB には接続しません。

専用 ProgID を使って MATLAB への接続を要求するクライアントは、別々の MATLAB を起動し、サーバは他のクライアントと共有されません。そのため、専用サーバは、複数のクライアントによって共有されないので、同時に複数の専用サーバを起動できます。

MATLAB を DCOM Server として使う

DCOM は、ネットワーク上で ActiveX の接続を行うプロトコルです。DCOM をサポートするバージョンの Windows(本マニュアル記述時点では Windows NT 4.0) と DCOM をサポートするコントローラを使っている場合は、リモートマシンで MATLAB を起動するためにコントローラを使うことができます。これを行うためには、DCOM は適切に設定され、クライアントまたはサーバとして使われる各マシンに MATLAB がインストールされなければなりません (クライアントマシンがそのような設定で MATLAB を実行していなくても、リモート接続を行うために MATLAB のコンポーネントが必要なので、クライアントマシンに MATLAB がインストールされていなければなりません)。ユーザの環境での DCOM の設定方法については、DCOM のドキュメントを参照してください。

Dynamic Data Exchange (DDE)

MATLAB は、他の Windows アプリケーションにアクセスしたり、他の Windows アプリケーションが MATLAB に幅広い状況でアクセスするための関数を提供します。これらの関数は、Microsoft Windows のアプリケーションがデータを交換して互いに通信するためのソフトウェアである dynamic data exchange (DDE) を使います。

本節では、MATLAB での DDE の利用について説明します。

- DDE の概念と用語
- MATLAB にサーバとしてアクセスする
- MATLAB をクライアントとして使う
- DDE Advisory リンク

DDE の概念と用語

アプリケーションは、DDE *通信*を行うことでお互いに通信を行います。通信を始めるアプリケーションは、クライアント と呼ばれます。クライアントアプリケーションに応答するアプリケーションは、*サーバ* と呼ばれます。

クライアントアプリケーションが DDE 通信を開始するとき、サーバで定義される 2 つの DDE パラメータを識別しなければなりません。

- サービス名 と呼ばれる通信を行おうとする *アプリケーション名*。
- トピック と呼ばれる通信の題名。

サーバアプリケーションがサポートされているトピックを含む通信の要求を受け取るとき、要求を認識し DDE 通信を確立します。サービスとトピックの組合せは、通信を一意的に識別します。サービスまたはトピックは、サービスが複数の通信を保持していても通信中に変更されません。

DDE 通信中に、クライアントとサーバのアプリケーションはアイテムに関係するデータを交換します。*アイテム* は、通信内の両方のアプリケーションにとって意味があるデータへの参照です。どちらのアプリケーションも、通信中にアイテムを変更できます。これらの概念は、下記で詳しく説明します。

サービス名

DDE サーバになりうるすべてのアプリケーションは、一意的なサービス名 を持ちます。サービス名は、普通アプリケーションの拡張子が付かない実行ファイル

名です。サービス名は、大文字と小文字の区別を行いません。以下は、一般に用いられるサービス名です。

- MATLAB に対するサービス名は *Matlab* です。
- Microsoft Word に対するサービス名は *WinWord* です。
- Microsoft Excel に対するサービス名は *Excel* です。

その他の Windows アプリケーションに対するサービス名は、アプリケーションのドキュメントを参照してください。

トピック

トピック は、DDE 通信の題名を定義し、普通クライアントとサーバのアプリケーションの両方に意味があるものです。トピック名は、大文字と小文字の区別を行いません。MATLAB のトピックは、System と Engine で、「サーバとして MATLAB にアクセスする」で説明します。ほとんどのアプリケーションは、System トピックと、少なくともあと 1 つのトピックをサポートします。サポートされているトピックについての情報は、アプリケーションのドキュメントを調べてください。

アイテム

各トピックは、1 つまたは複数のアイテムをサポートします。アイテムは、DDE 通信中に渡されるデータのことです。アイテムが大文字と小文字の区別を行うかどうかは、アプリケーションに依存します。行列名は区別を行うので、MATLAB Engine アイテムが行列を参照する場合は、区別を行います。

クリップボード書式

DDE は、アプリケーション間でフォーマットされたデータを送信するために、Windows のクリップボード書式を使います。クライアントとしては、MATLAB は Text 書式のみをサポートします。サーバとしては、MATLAB は下記に示すように Text, Metafilepict, XLTable 書式をサポートします。

- **Text** - Text 書式のデータは、null キャラクタで終わるキャラクタのバッファです。バッファ内のテキスト行は、キャリッジリターンとラインフィードの組み合わせで区切られます。バッファがデータの列を含む場合は、それらの列はタブキャラクタで区切られます。MATLAB は、リモートの EvalString コマンドの結果を得るために Text 書式をサポートし、行列データを要求します。行列データは、MATLAB に Text 書式で送信されます。
- **Metafilepict** - Metafilepict 書式は、グラフィックスの描画コマンドを含むグラフィカルデータの記述です。結果として、この書式で保存されたデータは、スケーリングが可能でデバイスに依存しません。MATLAB は、グラフィックの操

作を発生させるリモートコマンドの結果を得るためにMetafilepict書式をサポートします。

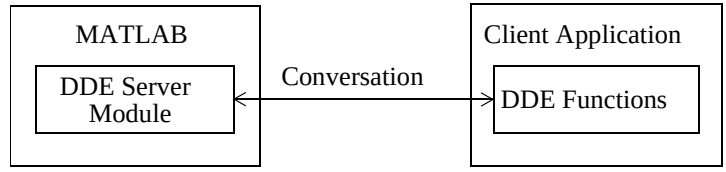
- **XLTable** - XLTable 書式は、Microsoft Excel で使われるクリップボード書式で、Excel とのデータ交換を効率良く簡単に行うためにサポートされます、XLTable 書式は、バッファ内に保持されるデータを記述したヘッダをもつバイナリのバッファです。XLTable 書式の詳しい説明は、Microsoft Excel SDK ドキュメンテーションを調べてください。

サーバとして MATLAB にアクセスする

クライアントのアプリケーションは、その種類に依存するつぎの方法で、DDE サーバとして MATLAB にアクセスできます。

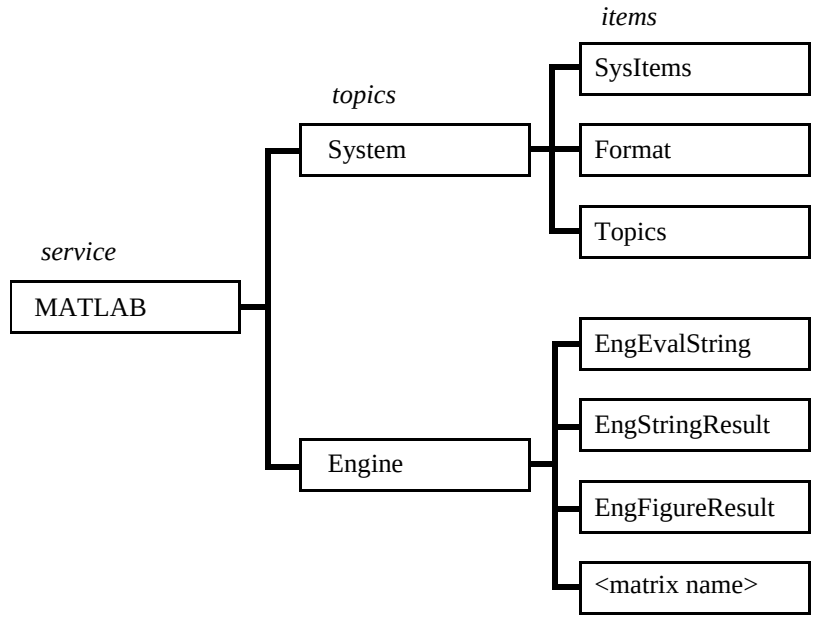
- DDE 通信を処理するために関数またはマクロを提供するアプリケーションを使っている場合は、これらの関数またはマクロを使うことができます。たとえば、Microsoft Excel, Word for Windows, Visual Basic は、DDE 関数またはマクロを提供します。これらの関数やマクロの使用の詳細は、適切な Microsoft のドキュメントを参照してください。
- ユーザアプリケーションを作成している場合は、MATLAB エンジンライブラリまたは DDE を直接使うことができます。エンジンライブラリの利用の詳細は、"MATLABエンジンを使って"MATLABエンジンを使ってを参照してください。DDE ルーチンの利用の詳細は、*Microsoft Windows Programmer's Guide* を参照してください。

下図は、MATLAB がサーバとしてどのように通信を行うかを示します。クライアントアプリケーションの DDE 関数は、MATLAB の DDE サーバモジュールと通信します。クライアントの DDE 関数は、アプリケーションまたは MATLAB エンジンライブラリにより提供されます。



DDE 名の階層

サーバとして MATLAB にアクセスするとき、サービス名、トピック、アイテムを指定しなければなりません。下図は、MATLAB DDE 名の階層を示したものです。トピックとアイテムについては、以下に詳細に記述されています。



2 つの MATLAB トピックは、*System* と *Engine* です。

MATLAB System トピック

System トピックを使って、サーバにより提供されるトピックのリスト、サーバにより提供される System トピックのアイテムのリスト、サーバによりサポートされる書式をブラウズすることができます。

MATLAB System トピックは、つぎのアイテムをサポートします。

- **SysItems**

  System トピックのもとでサポートされるアイテムのタブ分離したリストを提供します (このリストです)。

- **Format**

  サーバによりサポートされるすべての書式の文字列名のタブ分離したリストを提供します。MATLAB は、Text, Metafilepict, XLTable をサポートします。これらの書式は、「クリップボード書式」に説明されています。

- **Topics**

  MATLAB がサポートするトピック名のタブ分離したリストを提供します。

MATLAB Engine トピック

Engine トピックを使って、実行するコマンドに渡したり、データを要求、またはデータを送信することにより、MATLAB をサーバとして使うことができます。

MATLAB Engine トピックは、以下のアイテムをサポートします。

- **EngEvalString**

  評価のために MATLAB にコマンドを送信したときに、必要ならばアイテム名を指定します。

- **EngStringResult**

  MATLAB からデータを要求したときに DDE 実行コマンドの結果の文字列を与えます。

- **EngFigureResult**

  MATLABからデータを要求したときに、DDE実行コマンドのグラフィカルな結果を与えます。

- **<matrix name>**

  MATLAB からデータを要求したときに、データが要求される行列名です。MATLAB にデータを送信するときは、作成または更新される行列名です。

MATLAB Engineトピックは、DDEクライアントインタフェースをもつアプリケーションによって利用される 3 つのオペレーションをサポートします。これらのオペレーションは、評価のための MATLAB へのコマンドの送信、MATLAB からのデータの要求、MATLAB へのデータの送信を含みます。

評価のための MATLAB へのコマンドの送信

クライアントは、DDE execute オペレーションを使って MATLAB にコマンドを送信します。アイテム名と実行コマンドの指定を要求するクライアントと、コマンドの指定のみを要求するクライアントの 2 種類があるので、Engine トピックは、2 つの形式で DDE execute をサポートします。アイテム名が要求されるときは、EngEvalString を使ってください。両方の形式で、コマンドの書式は Text でなけれ

ばなりません。ほとんどのクライアントは、DDE execute に対して Text をデフォルトにします。書式が指定されなければ、おそらく Text です。以下の表は、DDE execute パラメータのまとめを示したものです。

アイテム	書式	コマンド
EngEvalString	Text	String
null	Text	String

MATLAB からのデータの要求

クライアントは、DDE request オペレーションを使って MATLAB からデータを要求します。Engineのトピックは、3つの機能に対するDDEの要求をサポートします。

前の DDE の実行コマンドの結果であるテキスト

Text書式でEngStringResultアイテムを使って、DDE実行コマンドの文字列の結果を要求します。

前の DDE 実行コマンドのグラフィックの結果

EngFigureResultアイテムを使ってDDE実行コマンドのグラフィックの結果を要求します。EngFigureResult アイテムは、Text または Metafilepict 書式で利用されます。

- Text 書式を指定すると、結果は "yes" または "no" の値をもつ文字列になります。結果が "yes" ならば、カレントの figure に対する metafile は、クリップボードに置かれます。この機能は、Word for Windows のように DDE 要求からテキストのみを読み込む DDE クライアントに対して提供されます。結果が "no" ならば、metafile はクリップボードに置かれません。
- グラフィックスの結果があるときにMetafilepict書式を指定すると、metafileは直接 DDE 要求から出力されます。

指定した行列に対するデータ

行列名をアイテムとして指定して、行列に対するデータを要求します。Text または XLTable 書式を指定することができます。

つぎの表は、DDE request パラメータのまとめです。

アイテム	書式	結果
EngStringResult	Text	String
EngFigureResult	Text	Yes/No
EngFigureResult	Metafilepict	カレント figure の Metafile
<*matrix name*>	Text	キャラクタバッファ、タブ分離された列、CR/LF 分離された行
<*matrix name*>	XLTable	Microsoft Excel 互換書式のバイナリファイル

MATLAB へのデータの送信

クライアントは、DDE poke オペレーションを使って MATLAB にデータを送信します。Engine トピックは、MATLAB ワークスペースでのアップデートや新しい行列の作成に対する DDE poke をサポートします。指定されるアイテムは、アップデートまたは作成する行列名です。指定した名前をもつ行列が既にワークスペースに存在していれば、アップデートされます。そうでない場合は、行列が作成されます。行列のデータは、Text または XLTable 書式です。

つぎの表は、DDE poke パラメータのまとめです。

アイテム	書式	Poke データ
<*matrix name*>	Text	キャラクタバッファ、タブ分離された列、CR/LF 分離された行
<*matrix name*>	XLTable	Microsoft Excel 互換書式のバイナリデータ

例題： Visual Basic と MATLAB DDE Server を使って

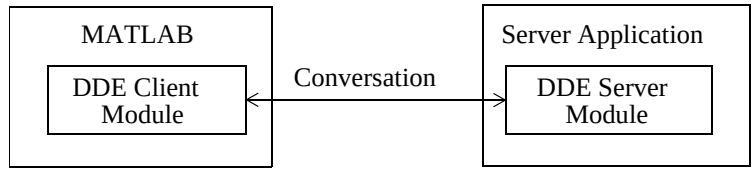
この例は、2 つのテキスト編集コントロール **TextInput** と **TextOutput** をもつ Visual Basic の形式です。このコードは、TextInput_KeyPress メソッドです。

```
Sub TextInput_KeyPress(KeyAscii As Integer)
rem If the user presses the return key
rem in the TextInput control.
If KeyAscii = vbKeyReturn then

rem Initiate the conversation between the TextInput
rem control and MATLAB under the Engine topic.
rem Set the item to EngEvalString.
                    TextInput.LinkMode = vbLinkNone
                    TextInput.LinkTopic = "MATLAB|Engine"
                    TextInput.LinkItem = "EngEvalString"
                    TextInput.LinkMode = vbLinkManual

rem Get the current string in the TextInput control.
rem This text is the command string to send to MATLAB.
                    szCommand = TextInput.Text

rem Perform DDE Execute with the command string.
                    TextInput.LinkExecute szCommand
                    TextInput.LinkMode = vbLinkNone

rem Initiate the conversation between the TextOutput
rem control and MATLAB under the Engine topic.
rem Set the item to EngStringResult.
                    TextOutput.LinkMode = vbLinkNone
                    TextOutput.LinkTopic = "MATLAB|Engine"
                    TextOutput.LinkItem = "EngStringResult"
                    TextOutput.LinkMode = vbLinkManual

rem Request the string result of the previous EngEvalString
rem command. The string ends up in the text field of the
rem control TextOutput.text.
                    TextOutput.LinkRequest
                    TextOutput.LinkMode = vbLinkNone

End If
End Sub
```

MATLAB をクライアントとして使う

MATLAB がクライアントのアプリケーションとして動作するためには、通信を作成し保持するために MATLAB DDE クライアント関数を使うことができます。

下図に、MATLAB がクライアントとしてサーバアプリケーションとどのように通信を行うかを示します。



MATLAB の DDE クライアントモジュールは、関数を含みます。以下の表に、MATLAB をクライアントとして使用できる関数を示します。

関数	説明
ddeadv	MATLAB と DDE サーバアプリケーション間で advisory リンクを設定します。
ddeexec	DDE サーバアプリケーションに実行文字列を送ります。
ddeinit	MATLAB と他のアプリケーション間で DDE 通信を開始します。
ddepoke	MATLAB から DDE サーバアプリケーションにデータを送ります。
ddereq	DDE サーバアプリケーションからデータを要求します。
ddeterm	MATLAB とサーバアプリケーション間の DDE 通信を終了します。
ddeunadv	MATLAB と DDE サーバアプリケーション間の advisory リンクを開放します。

サーバのアプリケーションが Microsoft Excel ならば、System のトピックまたはファイル名であるトピックを指定できます。後者を指定する場合、ファイル名は

.XLS または .XLC で終わり、必要ならば絶対パスを含みます。Microsoft Excel のアイテムはセルの参照で、個々のセルまたはセルの領域を参照できます。

Microsoft Word for Windows のトピックは、System と .DOC または .DOT で終わるファイル名のファイルに保存されるドキュメント名です。Word for Windows のアイテムは、トピックで指定されるドキュメントのブックマークです。

つぎの例は、Microsoft Excel との DDE 通信を作り、Excel に 20 行 20 列の行列のデータを送る M- ファイルです。

```
% Excel との通信を初期化
chan = ddeinit('excel', 'Sheet1');

%peaks プロットのサーフェスを作成
h = surf(peaks(20));
% サーフェスの z データを取得
z = get(h, 'zdata');

% Excel 内のセル範囲を設定
range = 'r1c1:r20c20';

% z データを Excel ワークシートへ挿入
rc = ddepoke(chan, range, z);
```

DDE Advisory リンク

サーバにあるデータが変更されるときに、クライアントアプリケーションに通知するために DDE を使うことができます。たとえば、Excel のスプレッドシートに入力されたデータを解析するために MATLAB を使う場合は、このデータが変更されるときに Excel が MATLAB に通知するためのリンクを作ることができます。また、新規のあるいは修正したスプレッドシートのデータを使って行列を自動的にアップデートするリンクを作ることができます。

アイテムの題名であるデータがサーバで変更されるときに、サーバアプリケーションが MATLAB に通知する方法により区別される 2 種類の advisory リンクを MATLAB はサポートします。

- アイテムによって定義されたデータが変更されるときにサーバが MATLAB にデータ名を与える *hot リンク*。
- データが変更されるときにサーバがMATLABに通知しますが、MATLABが要求するときにのみデータを与える *warm リンク* 。

関数 ddeadv と ddeunadv を使って advisory リンクのセットアップや開放を行います。MATLAB は、MATLAB がクライアントであるときにのみリンクをサポートします。

つぎの例は、クライアントとして振る舞う MATLAB と Microsoft Excel の間で DDE 通信を作ります。この例は、Excel と hot リンクを作ることで前の節の例を拡張したものです。リンクは行列 z をアップデートし、セル領域が変更されるときにコールバックを実行します。ユーザインタフェースコントロールであるプッシュボタンが押されると、advisory リンクと DDE 通信を終了します (グラフィカルユーザインタフェースの作成についての情報の詳細は、MATLAB マニュアル *Building GUIs with MATLAB* を参照してください)。

```
% Excel との通信を初期化
chan = ddeinit('excel', 'Sheet1');

% Excel 内のセル範囲を設定
range = 'r1c1:r20c20';

% peaks プロットのサーフェスを作成
h = surf(peaks(20));

% サーフェスの z データを取得
z = get(h, 'zdata');

% Excel スプレットシートに z データを挿入
rc = ddepoke(chan, range, z);

% Excel と MATLAB 行列 'z' との
% ホットリンク ADVISE をセットアップ
% コールバックは、zdata と cdata をサーフェス h が、
% Excel から新しいデータとなるように設定
rc = ddeadv(chan, range,...
    'set(h,"zdata",z);set(h,"cdata",z);','z');

% ADVISE リンクを終了し、DDE 通信を終了し、
% figure ウィンドウを閉じるための
% Push ボタンを作成
c = uicontrol('String','&Close','Position',[5 5 80 30],...
    'Callback',...
    'rc = ddeunadv(chan,range);ddeterm(chan);close;');
```

8

シリアルポート I/O

はじめに

MATLAB のシリアルポートインタフェースとは何か？

MATLAB のシリアルポートインタフェースは、コンピュータのシリアルポートに接続するモデム、プリンタ、科学機器のような周辺機器への直接のアクセスを提供します。このインタフェースは、シリアルポートオブジェクトによって確立されます。シリアルポートオブジェクトは、以下の実行を可能にする関数およびプロパティをサポートします。

- シリアルポート通信の設定
- シリアルポートコントロールピンの利用
- データの読み書き
- イベントとアクションの利用
- 情報をディスクに記録

多機能 I/O ボードのような PC 互換のデータ取得ハードウェアとの通信を行いたい場合は、Data Acquisition Toolbox が必要です。GPIB- または VISA 互換機器との通信を行いたい場合は、Instrument Control Toolbox が必要です。

これらのプロダクトに関する詳細は、MathWorks Web サイト、http://www.mathworks.com/products をご覧になってください。

サポートされるシリアルポートインタフェースの標準

長年にわたって、シリアルポートインタフェースの標準が開発されてきました。これらの標準には、RS-232, RS-422, RS-485 が含まれ、これらはすべて MATLAB のシリアルポートオブジェクトでサポートされます。これらのうちで、コンピュータと周辺機器との接続に最も広く用いられているインタフェースの標準は、RS-232 です。

本マニュアルでは「シリアルポートの概要」で説明している RS-232 標準を利用されていると仮定しています。利用しているインタフェース標準が何かは、コンピュータや機器のドキュメントを参照してください。

サポートされるプラットフォーム

MATLAB のシリアルポートインタフェースは、Microsoft Windows 95, Windows 98, Windows 2000, Windows NT, Linux, Sun Solaris プラットフォームに対してサポートされます。

ユーザデバイスでの例題の使用法

本節の多くの例題は、PCシリアルポートに接続された特定の周辺機器、特にCOM1ポートに接続されている Tektronix TDS 210 2 チャンネルオシロスコープを考慮しています。そのため、文字列コマンドの多くは、この機器に固有です。

周辺機器が他のシリアルポートに接続されている場合、あるいは、他のコマンドを受け付ける場合は、例題を修正してください。

シリアルポートの概要

本節では、以下のトピックスを使ってシリアルポートの概要を説明します。

- シリアル通信とは？
- シリアルポートインタフェース標準
- シリアルケーブルを使った 2 つのデバイスの接続
- シリアルポートの信号とピンの割り当て
- シリアルデータフォーマット
- プラットフォームに対応するシリアルポートの情報

多くのシリアルポートアプリケーションに対して、シリアルポートの動作についてあまり詳しくなくても、デバイスとコンピュータを接続し、データのやり取りを行うことが可能です。アプリケーションが単純な場合、あるいは上記のトピックスについて既に精通している場合は、「シリアルポートセッション」から始めて、MATLAB でのシリアルポートデバイスの使用法を見てください。

シリアル通信とは？

シリアル通信は、複数のデバイス間の通信のための最も一般的な低レベルプロトコルです。通常、1 つのデバイスがコンピュータで、それ以外のデバイスがモデム、プリンタ、別のコンピュータ、オシロスコープやファンクションジェネレータのような科学機器です。

その名が示す通り、シリアルポートは一度に 1 ビット、直列に情報を送受信します。これらのバイトは、バイナリ (数値) フォーマットまたはテキストフォーマットを使って送信されます。

シリアルポートインタフェース標準

2 つのデバイスの接続用のシリアルポートインタフェースは、Telecommunications Industry Association が発行する TIA/EIA-232C standard で指定されています。

オリジナルのシリアルポートインタフェース標準は、Recommended Standard number 232 を意味する RS-232 によって与えられます。用語 "RS-232" が一般的で、TIA/EIA-232 標準の次のシリアル通信ポートを参照するときに本マニュアルで利用されます。

RS-232 は、つぎのシリアルポートの特性を定義します。

- 最大ビット送信レートとケーブルの長さ
- 信号の名前、電気特性、機能
- 機械的な接続とピンの割り当て

主要な通信は、Transmit Data ピン、Receive Data ピン、Ground ピンの 3 本のピンを使って行われます。その他のピンは、データフロー制御に対しては可能ですが、必要ではありません。

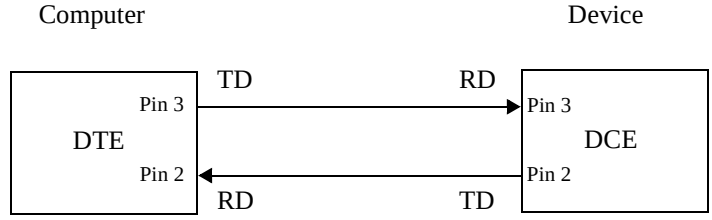
RS-485 のようなその他の標準は、より大きいビット送信レート、長いケーブル、最大 256 個までのデバイスの接続のような機能を定義します。

シリアルケーブルを使って 2 つのデバイスを接続

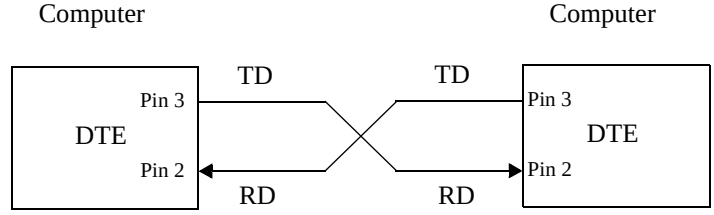
RS-232 標準は、シリアルケーブルで接続された 2 つのデバイスを Data Terminal Equipment (DTE) と Data Circuit-Terminating Equipment (DCE) として定義します。この用語は、コンピュータ端末とモデム間の通信用の標準として RS-232 を反映します。

本マニュアルでは、コンピュータは DTE で、モデムやプリンタのような周辺機器は DCE と考えます。多くの科学機器は、DTE として機能します。

RS-232 は主に DTE と DCE を接続することに関連するので、ピンの割り当ては、ストレートケーブルが利用されるように定義されます。ここで、pin 1 は、pin 1 に接続され、pin 2 は、pin 2 に接続されます。transmit data (TD) ピンと receive data (RD) ピンを使ったDTEとDCEのシリアル接続を以下に示します。シリアルポートのピンに関する情報は、「シリアルポートの信号とピンの割り当て」を参照してください。



2 つの DTE または 2 つの DCE をストレートシリアルケーブルを使って接続する場合は、各デバイスの TD ピンがお互いに接続され、各デバイスの RD ピンがお互いに接続されます。そのため、2 つの似ているデバイスを接続するには、null モデム ケーブルを使う必要があります。下記に示すように、null モデムケーブルは、ケーブルの送信ラインと受信ラインがクロスします。

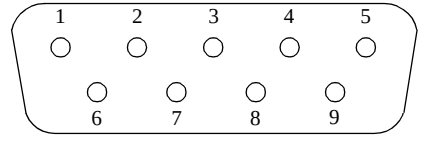


注意　　複数の RS-422 あるいは RS-485 デバイスをシリアルポートに接続することができます。RS-232/RS-485 アダプタをお持ちの場合は、これらのデバイスと共に MATLAB のシリアルポートオブジェクトを使うことができます。

シリアルポートの信号とピンの割り当て

シリアルポートは、2 つの信号タイプ、データ信号と制御信号で構成されます。これらの信号タイプと信号の接地をサポートするために、RS-232 標準は 25 ピン接続を定義します。しかし、ほとんどの PC および UNIX プラットフォームは、9 ピン接続を利用します。実際に、シリアルポート通信には、データ受信、データ送信、信号の接地用に 3 つのピンだけが必要です。

DTE での 9-p ピンオスコネクタに対するピンの割り当てを以下に示します。



9ピンコネクタに対応するピンと信号を以下に示します。25ピンコネクタで利用される信号とピンの割り当てについての説明は、RS-232 標準を参照してください。

表 8-1: シリアルポートのピンと信号の割り当て

ピン	ラベル	信号名	信号タイプ
1	CD	Carrier Detect	Control
2	RD	Received Data	Data
3	TD	Transmitted Data	Data
4	DTR	Data Terminal Ready	Control
5	GND	Signal Ground	Ground
6	DSR	Data Set Ready	Control
7	RTS	Request to Send	Control
8	CTS	Clear to Send	Control
9	RI	Ring Indicator	Control

用語 " データセット " は、" モデム " または " デバイス " と同義語で、" データターミナル " は " コンピュータ " と同義語です。

注意　シリアルポートのピンと信号の割り当ては、DTE に関するものです。たとえば、データは、DTE の TD ピンから DCE の RD ピンに送信されます。

信号の状態

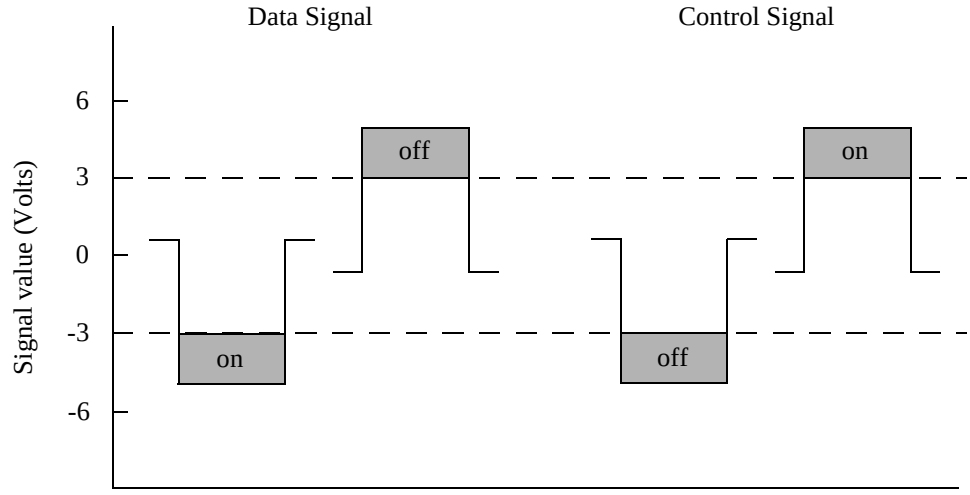
信号は、*アクティブ*な状態あるいは、*非アクティブ*な状態のいずれかです。アクティブな状態はバイナリ値 1 に対応し、非アクティブな状態はバイナリ値 0 に対応します。アクティブな信号の状態は、*logic 1, on, true, mark* として記述されます。非アクティブな信号の状態は、*logic 0, off, false, space* として記述されます。

データ信号に対して、"on" 状態は受信された信号電圧が -3 ボルトより低いときに発生し、"off" 状態は 3 ボルトよりも高いときに発生します。制御信号に対して、"on"状態は受信された信号電圧が3ボルトより高いときに発生し、"off"状態は-3ボ

ルトよりも低いときに発生します。-3 ボルトと +3 ボルトの間の電圧は遷移領域と考えられ、信号の状態は未定義です。

信号を "on" 状態にするには、制御デバイスはデータピンに対する値を unasserts（低く）し、制御ピンに対する値を asserts（高く）します。反対に、信号を "off" 状態にするには、制御デバイスはデータピンに対する値を高くし、制御ピンに対する値を低くします。

データ信号と制御信号に対する "on" および "off" 状態を以下に示します。



データピン

ほとんどのシリアルポートデバイスは、データを同時に送受信可能な*全二重*通信をサポートします。そのため、データの送信と受信には別々のピンが用いられます。これらのデバイスに対しては、TD, RD, GND ピンが用いられます。しかし、シリアルポートデバイスの中には、*一方向のみ通信*、あるいは*半二重通信*のみをサポートするものがあります。これらのデバイスに対しては、TD と GND ピンのみが用いられます。本マニュアルでは、全二重シリアルポートがデバイスに接続されていると仮定します。

TD ピンは、DTE から DCE に送信されたデータを運搬します。RD ピンは、DCE から DTE に受信されたデータを運搬します。

制御ピン

9 ピンシリアルポートは、以下を行う制御ピンを提供します。

- 接続されているデバイスの存在を知らせます。
- データフローを制御します。

制御ピンは、RTS と CTS, DTR と DSR, CD, RI を含みます。

RTS ピンと CTS ピン

RTS ピンと CTS ピンは、デバイスがデータの送受信のための準備ができているかどうかを通知するために用いられます。ハードウェアハンドシェイクと呼ばれるこのタイプのデータフロー制御は、送信中にデータが失われることを防ぐために用いられます。DTE と DCE の両方に対して利用可能なとき、RTS と CTS を利用したハードウェアハンドシェイクは、以下のステップを行います。

1. DTE は、RTS ピンをアサートして、データ受信の準備ができていることを DCE に通知します。
2. DCE は、TD ピン上でデータを送信することを示し CTS をアサートします。データが送信されない場合は、CTS ピンは unassert されます。
3. データは、TD ピン上で DTE に送信されます。データが受信されなくなった場合は、RTS ピンは DTE によって unassert され、データ送信は停止します。

MATLAB でハードウェアハンドシェイクを可能にするためには、「データフローの制御： ハンドシェイク」を参照してください。

DTR ピンと DSR ピン

多くのデバイスは、接続され電源が入っているかどうかを DSR ピンと DTR ピンを使って通知します。DTR と DSR を使って接続されているデバイスの存在を知らせるには、以下のステップを行います。

1. DTE は、DTR ピンをアサートして DCE が通信線を接続することを要求します。
2. DCE は、DSR ピンをアサートして接続されたことを通知します。
3. DCE は、DSR ピンが通信線から切断されたときに unassert します。

DTR ピンと DSR ピンは、他のハードウェアハンドシェイクの方法を提供するために元々は設計されました。しかし、RTS と CTS ピンは通常この方法で利用され、

DSR と DTR ピンは、そうではありません。しかし特定のピンの挙動を決定するためには、デバイスのドキュメントを参照するべきです。

CD ピンと RI ピン

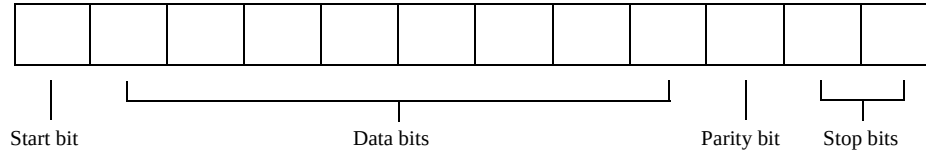
CD ピンと RI ピンは、モデム間の接続中の信号の存在を示すために利用されます。

CD は、他のモデムと接続していることを通知したり、キャリアトーンを検出したことを通知するためにモデムによって利用されます。CD は、DCE が適切な周波数の信号を受信中にアサートされます。CD は、DCE が適切な信号を受信していない場合には unassert されます。

RI は、電話のベルの音響信号の存在を示すために用いられます。RI は、ベルの信号を受信中のときにアサートされます。RI は、DCE がベルの信号を受信していないとき (例 ベルの間) に unassert されます。

シリアルデータフォーマット

シリアルデータフォーマットは、1 つのスタートビット、5 個から 8 個のデータビット、1 つのストップビットで構成されます。パリティビットとストップビットもフォーマットに含まれる場合があります。下記の図は、シリアルデータフォーマットを示したものです。



シリアルポートデータに対するフォーマットは、つぎの書式を使って表わされます。

number of data bits -Parity Type- number of stop bits

たとえば、8-N-1 は、8 個のデータビット、0 個のパリティビット、1 個のストップビットとして解釈されます。7-E-2 は 7 個のデータビット、偶数のパリティ、2 個のストップビットとして解釈されます。

これらのビットは通常 ASCII キャラクタを表わすため、データビットはキャラクタ として参照されることがあります。残りのビットはデータビットの枠なので、フレーミングビット と呼ばれます。

バイトと値

MATLAB では、フレーミングビットとデータビットは、まとめてバイト と呼ばれます。1 バイトは 8 ビットであり、シリアルデータフォーマットは 7 ビットから 12 ビットの範囲なので、最初はこの用語は不正確であると見られていました。しかし、シリアルデータがコンピュータに格納されると、フレーミングビットは取り除かれデータビットのみが保存されます。さらに、8 個のデータビットは、送信用に指定されたデータビット数に関わらず、常に未使用のビットを値 0 に割り当てて用いられます。

データの読み込みや書き出しのときに、1 つまたは複数のバイトからなる値 を指定する必要があることがあります。たとえば、int32 フォーマットを使ってデバイスから 1 つの値を読み込む場合は、その値は、4 個のバイトで構成されます。値の読み込みと書き出しに関する情報は、「データの書き出しと読み込み」を参照してください。

同期通信と非同期通信

RS-232 標準は、同期と非同期の 2 タイプの通信プロトコルをサポートします。

同期プロトコルを使って、送信されるすべてのビットは、共通のクロック信号に同期されます。2 つのデバイスは、最初はお互いに同期し、同期状態のままキャラクタを継続的に送信します。実際のデータが送信されていなくても、ビットの定流によって各デバイスが指定された時間に片方のデバイスがどこにあるかを知ることができます。つまり、送信される各ビットは実際のデータあるいはアイドルキャラクタです。各データバイトの先頭と末尾を示すビットが必要でないため、同期通信を使うと非同期法よりも速いデータ送信レートになります。

非同期プロトコルを使って、各デバイスは任意の時間に送信されるバイト単位の内部クロックを利用します。そのため、ビットの同期方法として時間を使う代わりにデータフォーマットが使われます。

特に、データ送信は、ワードのスタートビットを利用して同期化され、1 つまたは複数のストップビットがワードの末尾を示します。これらのビットを送信する必要があるため、非同期通信は同期通信よりもわずかに遅くなります。しかし、プロセッサがアイドルキャラクタを扱う必要がないという利点があります。ほとんどのシリアルポートは、非同期として動作します。

注意　本マニュアルで用いられる場合、用語 " 同期 " と " 非同期 " は、読み込みや書き出し操作が MATLAB コマンドラインへアクセスするかどうかを示します。詳細は、MATLAB コマンドラインへのアクセスの制御を参照してください。

ビットの送信方法？

定義によると、シリアルデータは、一度に 1 ビット送信されます。ビットの送信される順番を以下に示します。

1 スタートビットは、値 0 で送信されます。

2 データビットが送信されます。最初のデータビットは最下位のビット (LSB) に対応し、最後のデータビットは最上位のビット (MSB) に対応します。

3 パリティビット (定義されていると) が送信されます。

4 1 つまたは 2 つのストップビットが送信されます。各値は 1 です。

1 秒に送信されるビット数は、*ボーレート*によって与えられます。送信されるビットには、スタートビット、データビット、パリティビット (定義されていれば) およびストップビットが含まれます。

スタートビットとストップビット

同期通信と非同期通信シリアルデータフォーマットによるバイトの識別プロセスは、以下のステップに従います。シリアルデータフォーマットによるバイトの識別プロセスは、以下のステップに従います。

1 非同期シリアルポートピンがアイドル (データ送信中でない) のときは "on" 状態です。

2 データが送信されようとしているとき、シリアルポートピンはスタートビットによって "off" 状態に切り替えます。

3 シリアルポートピンは、ストップビットによって "on" 状態に戻ります。これは、バイトの終了を示します。

データビット

シリアルポートによって送信されるデータビットは、デバイスのコマンド、センサの読み込み、エラーメッセージ等を表わします。データは、バイナリデータまたは ASCII データとして送信されます。

ほとんどのシリアルポートは、5 ビットから 8 ビットを利用します。バイナリデータは、8 ビットとして送信されます。テキストベースのデータは、7 ビットまたは 8 ビットとして送信されます。データが ASCII キャラクタである場合は、2^7 または 128 個のキャラクタがあるため、最小でも 7 個のビットが必要です。8 ビット

目が利用される場合は、値 0 でなければなりません。データが拡張 ASCII キャラクタベースである場合は、2^8 個または 256 個のキャラクタがあるため、8 ビットを用いる必要があります。

パリティビット

パリティビットは、送信されたデータに対する簡単なエラー (パリティ) チェックを行います。パリティチェックのタイプを以下に示します。

表 8-2: パリティタイプ

パリティタイプ	説明
Even	データビットプラスパリティビットは、1 の数が偶数になります。
Mark	パリティビットは常に 1 です。
Odd	データビットプラスパリティビットは、1 の数が奇数になります。
Space	パリティビットは常に 0 です。

マークとスペースパリティチェックは、最小のエラー検出を行うのでほとんど使われません。パリティチェックを全く行わないことも選択可能です。

パリティチェックのプロセスは、以下のステップに従います。

1. 送信側のデバイスは、データビットの値および選択されたパリティチェックのタイプにより、パリティビットを 0 または 1 に設定します。
2. 受信側のデバイスは、パリティビットが送信データと一致するかどうかをチェックします。一致する場合はデータビットは受信されます。一致しない場合はエラーが出力されます。

注意　パリティチェックは、1 ビットエラーのみを検出可能です。複数ビットのエラーは有効なデータとして現れます。

たとえば、データビット 01110001 がコンピュータに送信されると仮定します。偶数のパリティが選択されると、パリティビットは送信デバイスによって 0 に設定

され、1 の数が偶数になります。奇数のパリティが選択されると、パリティビットは送信デバイスによって 1 に設定され、1 の数が奇数になります。

プラットフォームに対応するシリアルポートの情報

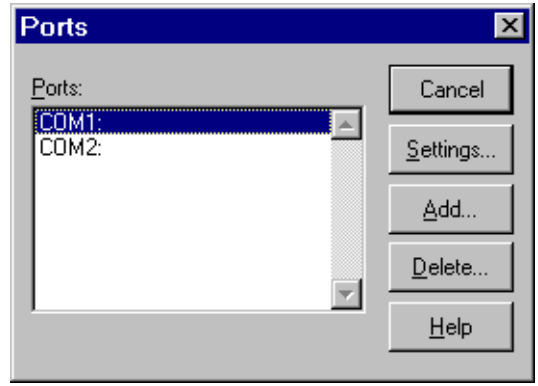
Windows および UNIX プラットフォームに対するシリアルポートの情報を見つける方法を以下に示します。

注意　オペレーティングシステムは、すべてのシリアルポートの設定に対するデフォルト値を提供します。しかし、これらの設定は、MATLAB コードにより上書きされるため、シリアルポートアプリケーションへの影響はありません。

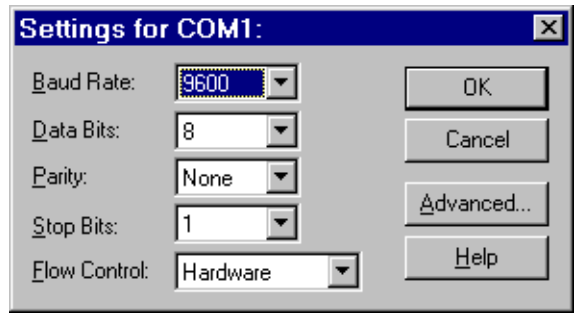
Windows プラットフォーム

Windows コントロールパネルを使ってシリアルポートの情報に簡単にアクセスすることができます。**スタート**ボタン (**スタート -> 設定 -> コントロールパネル**) を使ってコントロールパネルを呼び出すことができます。

Windows NT では、コントロールパネル内のポートアイコンを選択してシリアルポートにアクセスします。結果の**ポートダイアログボックス**は、以下のようになります。



COM1 で可能な設定について情報を得るには、ポートリストボックスで、この**ポート**を選択してから**設定**を選択します。



コントロールパネルから利用可能な **System Properties** ダイアログボックスを使って、Windows 95, Windows 98, Windows 2000 オペレーティングシステムに対するシリアルポートの情報にアクセスすることができます。

UNIX プラットフォーム

UNIXプラットフォームのシリアルポートの情報を得るためには、シリアルポート名を知る必要があります。これらの名前は、オペレーティングシステムにより異なります。

Linux では、シリアルポートデバイスは通常 ttyS0, ttyS1 等です。setserial コマンドを使ってシリアルポートの情報を表示あるいは設定することができます。たとえば、どのポートが利用可能かを表示するには、

```
setserial -bg /dev/ttyS*
/dev/ttyS0 at 0x03f8 (irq = 4) is a 16550A
/dev/ttyS1 at 0x02f8 (irq = 3) is a 16550A
```

ttyS0 に関する詳しい情報を表示するには

```
setserial -ag /dev/ttyS0
/dev/ttyS0, Line 0, UART: 16550A, Port: 0x03f8, IRQ: 4
    Baud_base: 115200, close_delay: 50, divisor: 0
    closing_wait: 3000, closing_wait2: infinte
    Flags: spd_normal skip_test session_lockout
```

注意　　setserial -ag コマンドが使えない場合は、ポートに対するリードとライトパーミッションがあることを確認してください。

サポートされているすべての UNIX プラットフォームに対して、stty コマンドを使ってシリアルポートの情報を表示あるいは設定することができます。たとえば、ttyS0 に対するシリアルポートプロパティを表示するには

```
stty -a < /dev/ttyS0
```

ボーレートを 4800 ビット / 秒に設定するには、

```
stty speed 4800 < /dev/ttyS0 > /dev/ttyS0
```

シリアル I/O の開始

MATLAB のシリアルポートインタフェースを開始するために、本節では以下の情報を説明します。

- " 例題：始めましょう " は、基本的なシリアルポートのコマンドを説明します。
- " シリアルポートセッション " は、シリアルポートのタスクを最初から最後まで行うために利用するステップを説明します。
- " プロパティの設定と出力 " は、シリアルポートプロパティ名と値の表示方法、プロパティへの値の割り当て方法を説明します。

例題： 始めましょう

シリアルポート COM1 に接続されていて、ボーレートが 4800 に設定されているデバイスをお持ちの場合は、つぎの例題を実行することができます。

```
s = serial('COM1');
set(s,'BaudRate',4800);
fopen(s);
fprintf(s,'*IDN?')
out = fscanf(s);
fclose(s)
delete(s)
clear s
```

*IDN? コマンドは、out に出力されるデバイスの識別情報を取得します。デバイスがこのコマンドを与えない場合、あるいは他のシリアルポートに接続されている場合は、上記の例題を修正してください。

注意　*IDN? は、Standard Commands for Programmable Instruments (SCPI) 言語によってサポートされているコマンドの 1 つで、多くのデバイスで利用されています。SCPI 言語をサポートするかどうかを確認するためにデバイスのドキュメントを参照してください。

シリアルポートセッション

シリアルポート *セッション* は、シリアルポートと接続しているデバイスとの通信時に行うすべてのステップで構成されます。ステップは、以下の通りです。

1 **シリアルポートオブジェクトの作成** - serial 関数を使って特定のシリアルポートに対してシリアルポートオブジェクトを作成します。

   オブジェクトの作成中にプロパティを設定することも可能です。特に、ボーレート、データビット数等のようなシリアルポート通信に関連するプロパティを設定したい場合があります

2 **デバイスの接続** - 関数fopenを使ってシリアルポートオブジェクトをデバイスに接続します。

   オブジェクトが接続された後で、プロパティ値を設定することによりデバイス設定を変更し、データを読み込み、データを書き出すことができます。

3 **プロパティの設定** - 希望するシリアルポートオブジェクトの挙動を確立するために、関数 set またはドット表記を使ってプロパティに値を割り当てます。

   実際には、オブジェクトの作成中、あるいは作成後にいつでも多くのプロパティを設定することが可能です。反対に、デバイス設定やシリアルポートアプリケーションの必要条件により、デフォルトのプロパティ値を受け取りこのステップを飛ばすことが可能です。

4 **データの書き出しと読み込み** - 関数 fprintf または fwrite を使ってデバイスにデータを書き出し、関数 fgetl, fgets, fread, fscanf, readasync を使ってデバイスからデータを読み込むことができます。

   シリアルポートオブジェクトは、あらかじめ設定した値またはデフォルトのプロパティ値に従って動作します。

5 **切断とクリーンアップ** - シリアルポートオブジェクトが必要なくなった場合は、関数 fclose を利用してデバイスから切断し、関数 delete を使ってメモリから削除し、clear コマンドを使って MATLAB ワークスペースから削除します。

シリアルポートセッションは、シリアルポートのドキュメントの多くの例題で補強されています。上記のステップを利用する基本的な例題は、8-18 ページの " 例題： 始めましょう " を参照してください。

プロパティの設定と出力

プロパティ値を設定することによって、希望するシリアルポートオブジェクトの挙動を設定します。関数 set、関数 get、ドット表記を使ってプロパティ値の表示や設定が可能です。

プロパティ名とプロパティ値の表示

シリアルポートオブジェクトが作成されると、関数 set を使ってすべての設定可能なプロパティをコマンドラインに表示することができます。さらに、プロパティが有限個の文字列の値をもつ場合は、set はこれらの値も表示します。

```
s = serial('COM1');
set(s)
    ByteOrder: [ {littleEndian} | bigEndian ]
    BytesAvailableAction
    BytesAvailableActionCount
    BytesAvailableActionMode: [ {terminator} | byte ]
    ErrorAction
    InputBufferSize
    Name
    OutputBufferSize
    OutputEmptyAction
    RecordDetail: [ {compact} | verbose ]
    RecordMode: [ {overwrite} | append | index ]
    RecordName
    Tag
    Timeout
    TimerAction
    TimerPeriod
    UserData

    SERIAL specific property
    BaudRate
    BreakInterruptAction
    DataBits
    DataTerminalReady: [ {on} | off ]
    FlowControl: [ {none} | hardware | software ]
    Parity: [ {none} | odd | even | mark | space ]
    PinStatusAction
    Port
    ReadAsyncMode: [ {continuous} | manual ]
```

```
RequestToSend: [ {on} | off ]
StopBits
Terminator: [ CR | {LF} | CR/LF | LF/CR ]
```

関数 get を使って 1 つまたは複数のプロパティとそれらのカレント値をコマンドラインに表示することができます。すべてのプロパティとそれらのカレント値を表示するには、以下のようにします。

```
get(s)
  ByteOrder = littleEndian
  BytesAvailable = 0
  BytesAvailableAction =
  BytesAvailableActionCount = 48
  BytesAvailableActionMode = terminator
  BytesToOutput = 0
  ErrorAction =
  InputBufferSize = 512
  Name = Serial-COM1
  OutputBufferSize = 512
  OutputEmptyAction =
  RecordDetail = compact
  RecordMode = overwrite
  RecordName = record.txt
  RecordStatus = off
  Status = closed
  Tag =
  Timeout = 10
  TimerAction =
  TimerPeriod = 1
  TransferStatus = idle
  Type = serial
  UserData = []
  ValuesReceived = 0
  ValuesSent = 0
  SERIAL specific property:
  BaudRate = 9600
  BreakInterruptAction =
  DataBits = 8
  DataTerminalReady = on
  FlowControl = none
  Parity = none
```

```
    PinStatus = [1x1 struct]
    PinStatusAction =
    Port = COM1
    ReadAsyncMode = continuous
    RequestToSend = on
    StopBits = 1
    Terminator = LF
```

1 つのプロパティに対するカレント値を表示するには、get でプロパティ名を指定します。

```
get(s,'OutputBufferSize')
ans =
  512
```

複数のプロパティに対するカレント値を表示するには、プロパティ名をセル配列の要素として指定する必要があります。

```
get(s,{'Parity','TransferStatus'})
ans =
    'none'    'idle'
```

ドット表記を使って 1 つのプロパティ値を表示することもできます。

```
s.Parity
ans =
none
```

プロパティ値の設定

関数 set を使ってプロパティ値を設定することができます。

```
set(s,'BaudRate',4800);
```

あるいは、ドット表記を使うことができます。

```
s.BaudRate = 4800;
```

複数のプロパティに対して値を設定するには、set で複数のプロパティ名とプロパティ値の組を指定します。

```
set(s,'DataBits',7,'Name','Test1-serial')
```

ドット表記では、一度に 1 つのプロパティ値しか設定できないことに注意してください。

実際には、シリアルポートオブジェクトが存在するときには、オブジェクトの生成中を含め、いつでも多くのプロパティを設定することが可能です。しかし、プロパティの中には、オブジェクトがデバイスに接続されているときあるいは情報をディスクに記録中には設定できないものがあります。プロパティがいつ設定可能かに関する情報は、8-71 ページの " プロパティリファレンス " を参照してください。

プロパティ名の指定

シリアルポートのプロパティ名は、大文字と小文字が混在して表わされます。これによりプロパティ名が読み易くなり、プロパティ名の指定時に任意の文字を使うことができます。さらに、プロパティを一意的に識別するために必要な文字のみを利用して、ほとんどのプロパティ名を短縮することが可能です。たとえば、つぎの方法で BaudRate プロパティを設定することができます。

```
set(s,'BaudRate',4800)
set(s,'baudrate',4800)
set(s,'BAUD',4800)
```

プロパティ名を M- ファイルで設定するときは、完全なプロパティ名を使う必要があります。これは、新規プロパティの追加により短縮名が一意でなくなる場合に、将来の MATLAB のリリースでの問題を予防します。

デフォルトのプロパティ値

プロパティに対して値を明示的に定義しない場合には、デフォルト値が利用されます。すべての設定可能なプロパティは、デフォルト値を持ちます。

注意　オペレーティングシステムは、ボーレートのようなすべてのシリアルポートの設定に対してデフォルト値を与えます。しかし、これらの設定は、ユーザの MATLAB コードによって変更され、シリアルポートアプリケーションに影響を与えません。

プロパティが有限個の文字列の値を持つ場合は、デフォルト値は {} で囲まれます。たとえば、Parity プロパティに対するデフォルト値は none です。

```
set(s,'Parity')
[ {none} | odd | even | mark | space ]
```

プロパティリファレンスページには、プロパティに対するデフォルト値が記述されています。

シリアルポートオブジェクトの作成

シリアルポートオブジェクトは、関数 serial を使って作成します。serial は、入力引数としてデバイスに接続されているシリアルポート名が必要です。さらに、オブジェクト生成中にプロパティ値を設定することができます。たとえば、シリアルポート COM1 に対応するシリアルポートオブジェクトを作成するには、以下のようにします。

```
s = serial('COM1');
```

シリアルポートオブジェクトは、MATLAB ワークスペースに存在します。whos コマンドを使って、s のクラスを表示することができます。

```
whos s
  Name      Size         Bytes  Class

  s         1x1            512  serial object

Grand total is 11 elements using 512 bytes
```

シリアルポートオブジェクトが作成されると、下記のプロパティに自動的に値を割り当てます。これらの汎用プロパティは、オブジェクトタイプとシリアルポートに基づいてシリアルポートオブジェクトに関する情報を提供します。

表 8-3: 汎用プロパティ

プロパティ名	説明
Name	シリアルポートオブジェクト名を指定します。
Port	プラットフォーム固有のシリアルポート名を示します。
Type	オブジェクトタイプを示します。

関数 get を使って、s に対するこれらのプロパティの値を表示することができます。

```
get(s,{'Name','Port','Type'})
ans =
    'Serial-COM1'    'COM1'    'serial'
```

オブジェクト作成中にプロパティを設定

オブジェクト作成中にシリアルポートのプロパティを設定することができます。serial は、プロパティ名とプロパティ値を関数 set と同じフォーマットで持ちます。たとえば、プロパティ名とプロパティ値の組を、つぎのようにして指定できます。

```
s = serial('COM1','BaudRate',4800,'Parity','even');
```

無効なプロパティ名を指定した場合は、オブジェクトは作成されません。しかし、複数のプロパティ（たとえば、BaudRate と Parity プロパティ）に対して無効な値を指定した場合は、オブジェクトは作成され、関数 fopen を使ってデバイスにオブジェクトを接続するまで、無効な値を通知しません。

シリアルポートオブジェクトの表示

シリアルポートオブジェクトは、重要な設定や状態の情報をまとめた便利な表示を提供します。つぎの 3 種類の方法でまとめの表示を行うことができます。

- コマンドラインでシリアルポートオブジェクト変数名を入力する。
- シリアルポートオブジェクト作成時にセミコロンを取り除く。
- ドット表記を使ってプロパティを設定時にセミコロンを取り除く。

シリアルポートオブジェクト s に対する表示のまとめを以下に示します。

```
Serial Port Object : Serial-COM1

Communication Settings
  Port:            COM1
  BaudRate:        9600
  Terminator:      LF

Communication State
  Status:          closed
  RecordStatus:    off

Read/Write State
  TransferStatus:  idle
  BytesAvailable:  0
  ValuesReceived:  0
  ValuesSent:      0
```

シリアルポートオブジェクトの配列を作成

MATLAB では、変数を繋げることによって既存の変数から配列を作成することができます。シリアルポートオブジェクトに対しても同様です。たとえば、シリアルポートオブジェクト s1 と s2 を作成するとします。

```
s1 = serial('COM1');
s2 = serial('COM2');
```

通常の MATLAB シンタックスを使って s1 と s2 からなるシリアルポートオブジェクト配列を作成することができます。列配列 x を作成するには、つぎのようにします。

```
x = [s1 s2]

Instrument Object Array

 Index:  Type:       Status:     Name:
 1       serial      closed      Serial-COM1
 2       serial      closed      Serial-COM2
```

行配列 y を作成するには、つぎのようにします。

```
y = [s1;s2];
```

シリアルポートオブジェクトの行列を作成することはできません。たとえば、つぎのような行列は作成できません。

```
z = [s1 s2;s1 s2];
??? Error using ==> serial/vertcat
Only a row or column vector of instrument objects can be created.
```

アプリケーションによっては、シリアルポートオブジェクトの配列を関数に渡したい場合があります。たとえば、set を 1 回呼び出して s1 と s2 に対してボーレートとパリティを設定するには、つぎのようにします。

```
set(x,'BaudRate',19200,'Parity','even')
```

その関数が入力としてシリアルポートオブジェクトを受け取るかを知るためには、シリアルポートの「関数リファレンス」を参照してください。

デバイスの接続

シリアルポートオブジェクトを使ってデータの読み書きを行う前に、関数 serial で指定したシリアルポートを通してデバイスに接続しなければなりません。関数 fopen を使ってシリアルポートオブジェクトをデバイスに接続します。

```
fopen(s)
```

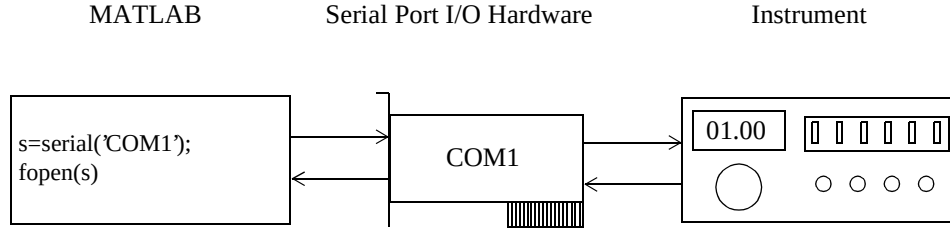
プロパティの中には、シリアルポートオブジェクトが接続されているときは参照のみで、fopen を使う前に設定しなければならないものがあります。例題は、InputBufferSize と OutputBufferSize プロパティを含みます。プロパティをいつ設定するかを決定するには、「プロパティリファレンス」を参照してください。

注意　任意数のシリアルポートを作成することが可能です。しかし、あるシリアルポートに一度に接続可能なシリアルポートオブジェクトは 1 つです。

シリアルポートオブジェクトがデバイスに接続されていることを確認するために、Status プロパティを調べることができます。

```
s.Status
ans =
open
```

下図のように、シリアルポートオブジェクトとデバイス間の接続が完了し、データの読み書きが可能になります。



通信プロパティの設定

データの読み書きを行う前に、シリアルポートオブジェクトとデバイスの両方が同一の通信設定でなければなりません。シリアルポート通信の設定は、ボーレートとシリアルデータフォーマットを制御するプロパティに対する値の指定を伴います。これらのプロパティは、以下の通りです。

表 8-4: 通信プロパティ

プロパティ名	説明
BaudRate	ビットが送信されるレートを指定します。
DataBits	送信するデータビット数を指定します。
Parity	パリティチェックのタイプを指定します。
StopBits	バイトの終了を示すビット数を指定します。
Terminator	ターミネータキャラクタを指定します。

注意　シリアルポートオブジェクトとデバイスの通信設定が異なる場合は、データの読み書きはできません。

サポートされている通信設定の説明は、デバイスのドキュメントを参照してください。

データの書き出しと読み込み

多くのシリアルポートアプリケーションに対して、データの読み書きを行うときに考慮すべき 3 つの重要な問題があります。

- 読み書きを行う関数は MATLAB コマンドラインへアクセスするか？
- 送信されるデータはバイナリ (数値) か、あるいはテキストか？
- どのような状況で読み書きの操作は終了するのか？

書き出しの操作に対するこれらの質問の回答は、8-31 ページ " データの書き出し " にあります。読み込み操作に対するこれらの質問の回答は、8-36 ページの " データの読み込み " にあります。

例題： データの書き出しと読み込み

シリアルポート COM1 に接続されている Tektronix TDS 210 two-channel oscilloscope の識別情報を出力したいとします。これは、関数 fprintf を使った *IDN? コマンドの機器への書き出しと、関数 fscanf を使った、そのコマンドの結果の読み込みが必要です。

```
s = serial('COM1');
fopen(s)
fprintf(s,'*IDN?')
out = fscanf(s)
```

結果の識別情報は、以下の通りです。

```
out =
TEKTRONIX,TDS 210,0,CF:91.1CT FV:v1.16 TDS2CM:CMV:v1.04
```

シリアルポートセッションを終了します。

```
fclose(s)
delete(s)
clear s
```

MATLAB コマンドラインへのアクセスの制御

読み込みあるいは書き出し操作が*同期* または*非同期* かどうかを指定することによって、MATLAB コマンドラインへのアクセスを制御することができます。

同期操作は、読み込みあるいは書き出し関数が実行を終了するまで、コマンドラインへのアクセスを防ぎます。非同期操作は、コマンドへのアクセスを制限せず、

読み込みや書き出し関数をバックグランドで実行中にコマンドを実行することができます。

用語 " 同期 " と " 非同期 " は、シリアルポートがハードウェアレベルでどのように動作するかを説明するときにしばしば用いられます。RS-232 標準は、非同期通信プロトコルをサポートします。このプロトコルを使って、各デバイスは独自の内部クロックを利用します。データ送信は、バイトのスタートビットを使って同期化され、1 つまたは複数のストップビットはバイトの終端を示します。スタートビットとストップビットに関する情報は、8-11 ページの " シリアルデータフォーマット " を参照してください。RS-232 標準は、送信されるすべてのビットが共有のクロック信号に同期化される同期モードもサポートします。

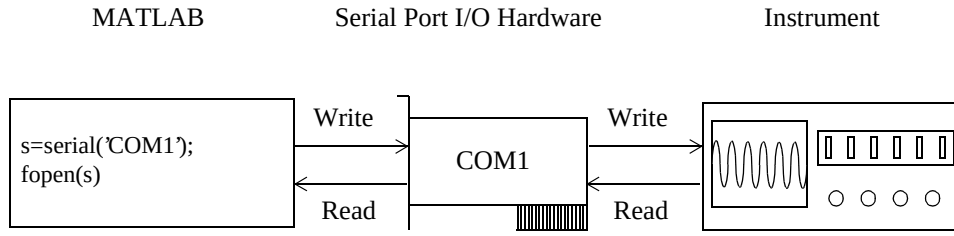
ハードウェアレベルでは、ほとんどのシリアルポートは非同期的に動作します。しかし、読み込みおよび書き出し関数に対するデフォルトの挙動を使って、同期シリアルポートの操作に似せることができます。

注意　本マニュアルで用いられるとき、用語 " 同期 " および " 非同期 " は、読み込みあるいは書き出し操作が MATLAB コマンドラインへのアクセスを防ぐかどうかを示します。言い換えると、これらの用語はハードウェアの挙動ではなくソフトウェアの挙動を意味します。

データを非同期で書き出しあるいは読み込むことには2つの主な利点があります。

- 書き出しまたは読み込み関数の実行中に他のコマンドを実行することができます。
- サポートされているすべてのアクションプロパティを利用することができます (8-48 ページの " イベントとアクションの利用 " を参照)。

たとえば、シリアルポートは別々の読み込み用と書き出し用のピンを持つため、データの読み込みと書き出しを同時に行うことができます。下図に示す通りです。



データの書き出し

本節は、データのシリアルポートデバイスへの書き出しを 3 つの部分に分けて説明します。

- "出力バッファとデータフロー"は、MATLABからデバイスへのデータの流れを説明します。
- "テキストデータの書き出し"は、テキストデータ(文字列コマンド)をデバイスに書き出す方法を説明します。
- "バイナリデータの書き出し"は、バイナリデータ(数値)をデバイスに書き出す方法を説明します。

データの書き出しに関連する関数は、以下の通りです。

表 8-5: データの書き出しに関連する関数

関数名	説明
fprintf	テキストをデバイスに書き出します。
fwrite	バイナリデータをデバイスに書き出します。
stopasync	非同期の読み込みおよび書き出し操作を終了します。

データの書き出しに関連するプロパティは、以下の通りです。

表 8-6: データの書き出しに関連するプロパティ

プロパティ名	説明
BytesToOutput	現在出力バッファにあるバイト数を示します。
OutputBufferSize	出力バッファのサイズをバイト単位で指定します。
Timeout	読み込みまたは書き出し操作を終了するまでの待ち時間を指定します。
TransferStatus	進行中の同期読み込みあるいは書き出し操作を示します。
ValuesSent	デバイスに書き出される値の総数を示します。

出力バッファとデータフロー

出力バッファは、デバイスに書き出されるデータを格納するためにシリアルポートオブジェクトによって割り当てられるコンピュータメモリです。データをデバイスに書き出す際に、データフローは、つぎの 2 つのステップに従います。

1 書き出し関数によって指定されたデータは、出力バッファに送られます。

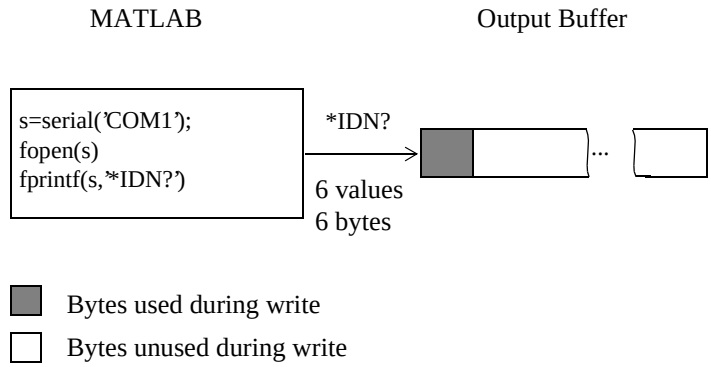
2 出力バッファ内のデータは、デバイスに送られます。

OutputBufferSize プロパティは、出力バッファに格納可能な最大バイト数を指定します。BytesToOutput プロパティは、出力バッファに現在あるバイト数を示します。これらのプロパティのデフォルト値は、以下のように与えられます。

```
s = serial('COM1');
get(s,{'OutputBufferSize','BytesToOutput'})
ans =
    [512]    [0]
```

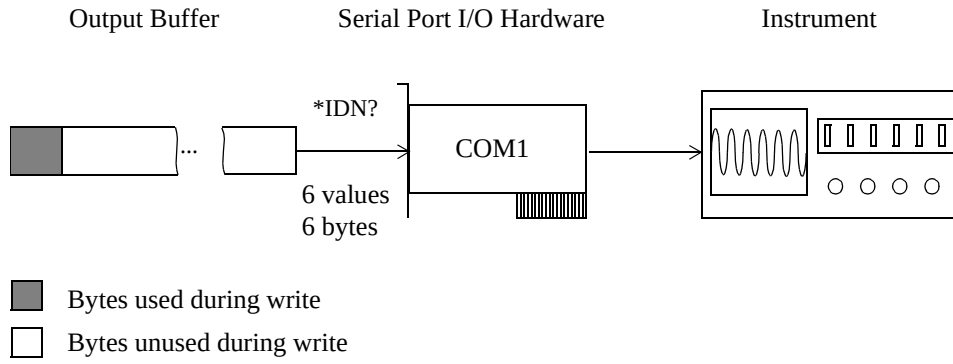
出力バッファの容量以上のデータを書こうとした場合は、エラーが出力され、データは書き出されません。

たとえば、関数 fprintf を使って文字列コマンド *IDN? を TDS 210 オシロスコープに書くと仮定します。下記のように、文字列は最初に 6 個の値として出力バッファに書き出されます。



*IDN? コマンドは、ターミネータが自動的に書き出されるため、6 個の値で構成されます。さらに、関数 fprintf に対するデフォルトのデータフォーマットは、1 つの値が 1 バイトに対応すると指定します。バイトと値に関する詳しい情報は、8-11 ページの " バイトと値 " を参照してください。fprintf とターミネータについては、8-34 ページの " テキストデータの書き出し " で説明しています。

下記のように、文字列は出力バッファに書き出された後で、シリアルポートによってデバイスに書き出されます。



テキストデータの書き出し

テキストデータをデバイスに書き出すためには、関数 fprintf を使います。多くのデバイスに対して、テキストデータの書き出しは、デバイスの設定を変更したり、データや状態の情報の出力のためにデバイスを準備する等のコマンド文字列の書き出しを意味します。

たとえば、Display:Contrast コマンドはオシロスコープの表示のコントラストを変更します。

```
s = serial('COM1');
fopen(s)
fprintf(s,'Display:Contrast 45')
```

デフォルトでは、fprintf は多くのシリアルポートデバイスがテキストベースのコマンドしか受け取らないため、%s\n フォーマットを使ってデータを書き出します。しかし、fprintf リファレンスページで説明するように、その他の多くのフォーマットを指定することが可能です。

デバイスに送信される値の数を ValuesSent プロパティを使って確認することができます。

```
s.ValuesSent
ans =
   20
```

ValuesSent プロパティの値は、デバイスに送信されるコマンド内の \n が Terminator プロパティの値で置き換えられるため、ターミネータを含みます。

```
s.Terminator
ans =
LF
```

Terminator のデフォルト値は、ラインフィードキャラクタです。デバイスに対して必要なターミネータは、デバイスのドキュメントに記述されています。

同期と非同期の書き込み操作

デフォルトでは、fprintf は同期的に実行されて実行が終了するまで MATLAB コマンドラインをブロックします。テキストデータを非同期的にデバイスに書き出すには、fprintf の最後の入力引数として async を指定する必要があります。

```
fprintf(s,'Display:Contrast 45','async')
```

非同期操作は、MATLAB コマンドラインへのアクセスをブロックしません。さらに、非同期の書き出し操作が進行中には、以下を行うことができます。

- シリアルポートは読み込みと書き出しに対して別々のピンを持つため、非同期の読み込み操作を行います。
- サポートされているすべてのアクションプロパティを利用します。

どの非同期操作が進行中かを TransferStatus プロパティを使って決定することができます。非同期操作が進行中でない場合は、TransferStatus は idle です。

```
s.TransferStatus
ans =
idle
```

fprintf による書き出し操作の終了の規則

fprintf を使った同期あるいは非同期の書き出し操作は、以下のときに終了します。

- 指定したデータが書き出されたとき。
- Timeout プロパティで指定した時間が経過したとき。

さらに、関数 stopasync を使って非同期操作を終了することができます。

バイナリデータの書き出し

バイナリデータをデバイスに書き出すためには、関数 fwrite を使います。バイナリデータを書き出すことは、数値を書き出すことを意味します。バイナリデータ

の書き出しの典型的なアプリケーションは、任意波形発生器のような機器にキャリブレーションデータを書き出すことです。

注意　シリアルポートデバイスの中には、テキストベースのコマンドのみを受け取るものがあります。これらのコマンドは、SCPI 言語あるいはベンダー固有の言語を利用する場合があります。そのため、すべての書き出し操作に対して関数 fprintf を使う必要があります。

デフォルトでは、fwrite は uchar フォーマットを使って値を変換します。しかし、この関数のリファレンスページで説明するように、他の多くのフォーマットを指定することが可能です。

デフォルトでは、fwrite は同期的に動作します。バイナリデータを非同期的にデバイスに書き出すには、fwrite の最終入力引数として async を指定する必要があります。同期および非同期の書き出し操作に関する詳しい情報は、テキストデータの書き出しを参照してください。書き出し操作を終了するために fwrite で用いる規則については、リファレンスページを参照してください。

データの読み込み

本節では、シリアルポートデバイスからのデータの読み込みについて 3 つの部分に分けて説明します。

- "入力バッファとデータフロー"は、デバイスから MATLAB へのデータフローを説明します。
- "テキストデータの読み込み"は、デバイスからの読み込み方法やデータをテキストとして書式化する方法を説明します。
- "バイナリデータの読み込み"は、デバイスからバイナリ (数値) データを読み込む方法を説明します。

データの読み込みに関連する関数は、以下の通りです。

表 8-7: データの読み込みに関連する関数

関数名	説明
fgetl	デバイスからテキストを 1 行読み込み、ターミネータを破棄します。
fgets	デバイスからテキストを 1 行読み込み、ターミネータを含めます。
fread	デバイスからバイナリデータを読み込みます。
fscanf	デバイスからデータを読み込み、テキストとして書式化します。
readasync	データをデバイスから非同期的に読み込みます。
stopasync	非同期読み込みと書き出し操作を終了します。

データの読み込みに関連するプロパティは、以下の通りです。

表 8-8: データの読み込みに関連するプロパティ

プロパティ名	説明
BytesAvailable	入力バッファ内で利用可能なバイト数を示します。
InputBufferSize	入力バッファのサイズをバイト単位で指定します。
ReadAsyncMode	同期読み込み操作が連続かマニュアルかを指定します。
Timeout	読み込みまたは書き込み操作を完了する待ち時間を指定します。
TransferStatus	進行中の非同期読み込みまたは書き出し操作を示します。
ValuesReceived	デバイスから読み込む値の総数を示します。

入力バッファとデータフロー

入力バッファは、デバイスから読み込まれるデータを格納するために、シリアルポートオブジェクトによって割り当てられるコンピュータメモリです。デバイスからデータを読み込む際に、データフローはつぎの 2 つのステップに従います。

1 デバイスから読み込まれるデータは、入力バッファに格納されます。

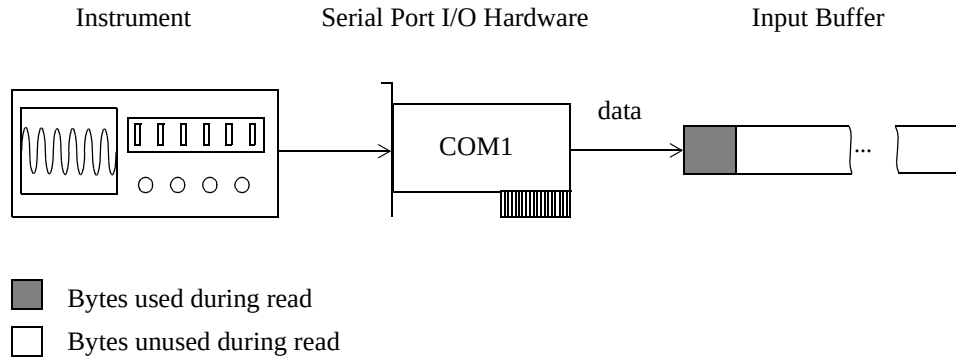
2 入力バッファ内のデータは、読み込み関数で指定されたMATLAB変数に出力されます。

InputBufferSizeプロパティは、入力バッファ内に格納可能な最大バイト数を指定します。BytesAvailable プロパティは、入力バッファから現在読み込み可能なバイト数を示します。これらのプロパティのデフォルト値は、以下のように与えられます。

```
s = serial('COM1');
get(s,{'InputBufferSize','BytesAvailable'})
ans =
    [512]    [0]
```

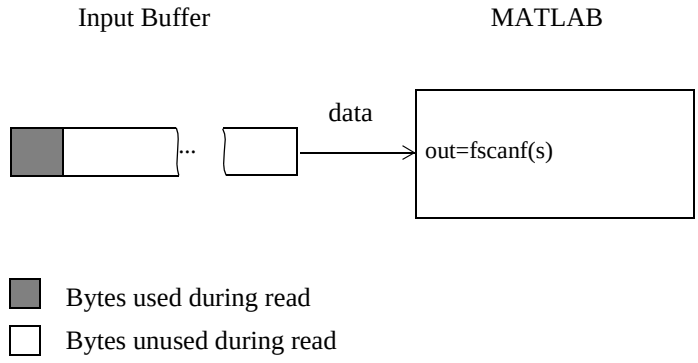
入力バッファの容量以上のデータを読み込もうとする場合は、エラーが出力され、データは読み込まれません。

たとえば、関数 fscanf を使って TDS 210 オシロスコープに書き出された *IDN? コマンドのテキストベースの応答を読み込みます。下記のように、テキストデータは、最初にシリアルポートから入力バッファに読み込まれます。



与えられた読み込み操作に対して、デバイスから出力されたバイト数がわからない場合があります。そのため、シリアルポートオブジェクトに接続する前に InputBufferSize プロパティを十分な大きさの値に設定する必要があります。

下記のように、データは入力バッファに格納された後で、fscanf によって指定された出力変数に送信されます。



テキストデータの読み込み

関数 fgetl, fgets, fscanf を使ってデバイスからデータを読み込み、データをテキストとして書式化することができます。

たとえば、オシロスコープに対する識別情報を出力したいと仮定します。これは、*IDN? コマンドを機器に書き出して、コマンドの結果を読み込む必要があります。

```
s = serial('COM1');
fopen(s)
fprintf(s,'*IDN?')
out = fscanf(s)
out =
TEKTRONIX,TDS 210,0,CF:91.1CT FV:v1.16 TDS2CM:CMV:v1.04
```

デフォルトでは、多くのシリアルポートデバイスから出力したデータはテキストベースなので、fscanf は %c フォーマットを使ってデータを読み込みます。しかし、fscanf リファレンスページで説明するように、その他の多くのフォーマットを指定することが可能です。

デバイスから読み込まれる値の数 (ターミネータを含む) を ValuesReceived プロパティを使って確認することができます。

```
s.ValuesReceived
ans =
   56
```

同期と非同期の読み込み操作

読み込み操作が同期かあるいは非同期であるかを ReadAsyncMode プロパティを使って指定します。ReadAsyncMode は、continuous または manual に設定することができます。

ReadAsyncModeがcontinuous(デフォルト値)の場合、シリアルポートオブジェクトは、データが読み込み可能かどうかを決定するために、デバイスに連続的に問い合わせます。データが読み込み可能の場合は、入力バッファに非同期的に格納されます。データを入力バッファから MATLAB へ送信するために、fgetl または fscanf のような同期 (blocking) 読み込み関数を使います。データが入力バッファで読み込み可能の場合は、これらの関数は素早く出力します。

```
s.ReadAsyncMode = 'continuous';
fprintf(s,'*IDN?')
s.BytesAvailable
ans =
   56
out = fscanf(s);
```

ReadAsyncMode が manual の場合、シリアルポートオブジェクトは、データが読み込み可能かどうかを決定するために連続的にデバイスに問い合わせません。その代わりに、読み込み関数を実行して入力バッファにマニュアルで入力しなければなりません。データを非同期的に読み込むには、関数 readasync を使います。その後、同期読み込み関数を使ってデータを入力バッファから MATLAB に送信します。

```
s.ReadAsyncMode = 'manual';
fprintf(s,'*IDN?')
s.BytesAvailable
ans =
   0
readasync(s)
s.BytesAvailable
ans =
   56
out = fscanf(s);
```

非同期操作は、MATLAB コマンドラインへのアクセスをブロックしません。さらに、非同期読み込み操作の進行中には、以下を行うことができます。

- シリアルポートは読み込みと書き出しに対して別々のピンを持つため、非同期の書き出し操作を実行します。
- サポートされているすべてのアクションプロパティを利用します。

どの非同期操作が進行中かを TransferStatus プロパティを使って決定することができます。非同期操作が進行中でない場合は、TransferStatus は idle です。

```
s.TransferStatus
ans =
idle
```

fscanf による読み込み操作の終了の規則

fscanf を使った読み込み操作は、以下が行われるときまで、MATLAB コマンドラインへのアクセスをブロックします。

- Terminator プロパティで指定されたターミネータが読み込まれるまで。
- Timeout プロパティで指定された時間が経過するまで。
- 指定された数の値が読み込まれるまで。
- 入力バッファが満たされるまで (値の数が指定されていない場合)。

バイナリデータの読み込み

バイナリデータをデバイスから読み込むためには、関数 fread を使います。バイナリデータを読み込むことは、数値を MATLAB に出力することを意味します。

たとえば、カーソルを出力し、オシロスコープに対する設定を表示したいとします。これは、CURSOR? および DISPLAY? コマンドを機器に書き出し、それらのコマンドの結果を読み込む必要があります。

```
s = serial('COM1');
fopen(s)
fprintf(s,'CURSOR?')
fprintf(s,'DISPLAY?')
```

ReadAsyncModeプロパティに対するデフォルト値はcontinuousなので、データはデバイスから読み込み可能になるとすぐに入力バッファに非同期的に出力されます。BytesAvailable プロパティを使って読み込まれる値の数を調べることができます。

```
s.BytesAvailable
ans =
   69
```

同期読み込み関数を使って MATLAB にデータを出力することができます。しかし、fgetl, fgets, fscanf を利用する場合は、入力バッファに 2 つのターミネータが格納されるため、関数を 2 回実行する必要があります。fread を利用する場合は、すべてのデータを 1 回の関数呼び出しで MATLAB に出力することができます。

```
out = fread(s,69);
```

デフォルトでは、fread は数値を倍精度配列に出力します。しかし、fread リファレンスページで説明するように他の多くのフォーマットを指定することが可能です。MATLAB関数charを使って数値データをテキストに変換することができます。

```
val = char(out)'
val =
HBARS;CH1;SECONDS;-1.0E-3;1.0E-3;VOLTS;-6.56E-1;6.24E-1
YT;DOTS;0;45
```

同期および非同期の読み込み操作に関する詳しい情報は、テキストデータの読み込みを参照してください。読み込み操作を終了するために fread で用いる規則については、リファレンスページを参照してください。

例題： テキストデータの書き出しと読み込み

この例題は、テキストデータを書き出しおよび読み込むことによって、シリアルポートの機器との通信の方法を説明します。

機器は、シリアルポート COM1 に接続されている Tektronix TDS 210 two-channel oscilloscope です。そのため、下記のコマンドの多くはこの機器に固有です。正弦波がオシロスコープのチャンネル2の入力であり、行うべきことは入力信号のピーク間電圧を測定することです。

1．シリアルポートオブジェクトを作成します。 - シリアルポート COM1 に対応するシリアルポートオブジェクト s を作成します。

```
s = serial('COM1');
```

2．デバイスを接続します。 - s をオシロスコープに接続します。ReadAsyncMode プロパティに対するデフォルト値は continuous なので、データは機器から利用可能になるとすぐに入力バッファに非同期的に出力されます。

```
fopen(s)
```

3．データの書き出しと読み込み - fprintf を使って *IDN? コマンドを機器に書き出し、fscanf を使ってコマンドの結果を読み込みます。

```
fprintf(s,'*IDN?')
idn = fscanf(s)
idn =
TEKTRONIX,TDS 210,0,CF:91.1CT FV:v1.16 TDS2CM:CMV:v1.04
```

測定のソースを決定する必要があります。可能な測定ソースには、オシロスコープの channel 1 と channel 2 が含まれます。

```
fprintf(s,'MEASUREMENT:IMMED:SOURCE?')
source = fscanf(s)
source =
CH1
```

チャンネル 1 からの測定結果を出力する範囲が設定されます。入力信号はチャンネル 2 に接続されるので、このチャンネルからの測定結果を出力する機器を設定しなければなりません。

```
fprintf(s,'MEASUREMENT:IMMED:SOURCE CH2')
fprintf(s,'MEASUREMENT:IMMED:SOURCE?')
source = fscanf(s)
source =
CH2
```

ピーク間電圧を出力する範囲を設定し、この測定結果の値を要求するように設定します。

```
fprintf(s,'MEASUREMENT:MEAS1:TYPE PK2PK')
fprintf(s,'MEASUREMENT:MEAS1:VALUE?')
```

fscanf を使って、入力バッファから MATLAB にデータを送信します。

```
ptop = fscanf(s,'%g')
ptop =
2.0199999809E0
```

4．切断とクリーンアップ - s が必要なくなったときは、機器から切断し、メモリおよび MATLAB ワークスペースから削除します。

```
fclose(s)
delete(s)
clear s
```

例題： strread を使ったデータの解釈

この例題は、関数 strread を使ってデバイスから読み込んだデータを書式化する方法を説明します。strread は、特に 1 つまたは複数の変数に文字列を構文解析したい場合に役立ちます。ここで、各変数は独自に指定されたフォーマットを持ちます。

機器は、シリアルポート COM1 に接続された Tektronix TDS 210 two-channel oscilloscope です。

1．シリアルポートオブジェクトを作成します。 - シリアルポート COM1 に対応するシリアルポートオブジェクト s を作成します。

```
s = serial('COM1');
```

2．デバイスを接続します。 - s をオシロスコープに接続します。ReadAsyncMode プロパティに対するデフォルト値は continuous なので、データは機器から利用可能になるとすぐに入力バッファに非同期的に出力されます。

```
fopen(s)
```

3．データの書き出しと読み込み - fprintf を使って RS232? コマンドを機器に書き出し、fscanf を使ってコマンドの結果を読み込みます。RS232? は、RS-232 の設定を質問し、ボーレート、ソフトウェアのフローコントロールの設定、ハードウェアのフローコントロールの設定、パリティタイプ、ターミネータを出力します。

```
fprintf(s,'RS232?')
data = fscanf(s)
data =
9600;0;0;NONE;LF
```

関数 strread を使ってデータ変数を構文解釈し、変数 data を 5 個の新規の変数に書式化します。

```
[baud,swfc,hwfc,par,term] = strread(data,'%d%d%d%s%s', 'delimiter',';')
baud =
      9600
swfc =
    0
hwfc =
    0
par =
   'NONE'
term =
   'LF'
```

4．切断とクリーンアップ - s が必要なくなったときは、機器から切断し、メモリおよび MATLAB ワークスペースから削除します。

```
fclose(s)
delete(s)
clear s
```

例題： バイナリデータの読み込み

この例題は、TDS 210 オシロスコープのスクリーン表示を MATLAB にどのようにダウンロードするかを説明します。スクリーン表示データは、Windows のビットマップフォーマットを使ってディスクに送信され、保存されます。このデータは、ユーザの作業の永久的な記録で、重要な信号や範囲のパラメータを記述するための簡単な方法です。

送信されるデータ量は、かなり大きいと予想されるので、機器から利用可能になるとすぐに入力バッファに非同期的に出力されます。これにより、他のタスクを送信が進行するにつれて実行することができます。さらに、範囲は最大のボーレート 19200 に設定されます。

1．シリアルポートオブジェクトを作成します。 - シリアルポート COM1 に対応するシリアルポートオブジェクト s を作成します。

```
s = serial('COM1');
```

2．プロパティ値を設定します。 - 入力バッファをかなり大きなバイト数を受け入れるように設定し、ボーレートを範囲によってサポートされる最大値に設定します。

```
s.InputBufferSize = 50000;
s.BaudRate = 19200;
```

3．デバイスに接続します。- s をオシロスコープに接続します。ReadAsyncMode プロパティに対するデフォルト値は continuous なので、データは機器から利用可能になるとすぐに入力バッファに非同期的に出力されます。

```
fopen(s)
```

4．データの書き出しと読み込み - スクリーン表示をビットマップとして送信する範囲を設定します。

```
fprintf(s,'HARDCOPY:PORT RS232')
fprintf(s,'HARDCOPY:FORMAT BMP')
fprintf(s,'HARDCOPY START')
```

すべてのデータが、入力バッファに送られるまで待って、その後、符号なしの 8-ビット整数として、MATLAB ワークスペースに転送します。

```
out = fread(s,s.BytesAvailable,'uint8');
```

5．切断とクリーンアップ - s が必要なくなったときは、機器から切断し、メモリおよび MATLAB ワークスペースから削除します。

```
fclose(s)
delete(s)
clear s
```

ビットマップデータの表示

ビットマップデータを表示するには、つぎのステップに従います。

1 ディスクファイルをオープンします。

2 データをディスクファイルに書き出します。

3 ディスクファイルをクローズします。

4 関数 imread を使ってデータを MATLAB に読み込みます。

5 関数 imagesc を使ってデータをスケーリングして表示します。

fopen, fwrite, fclose のファイル I/O バージョンが利用されます。

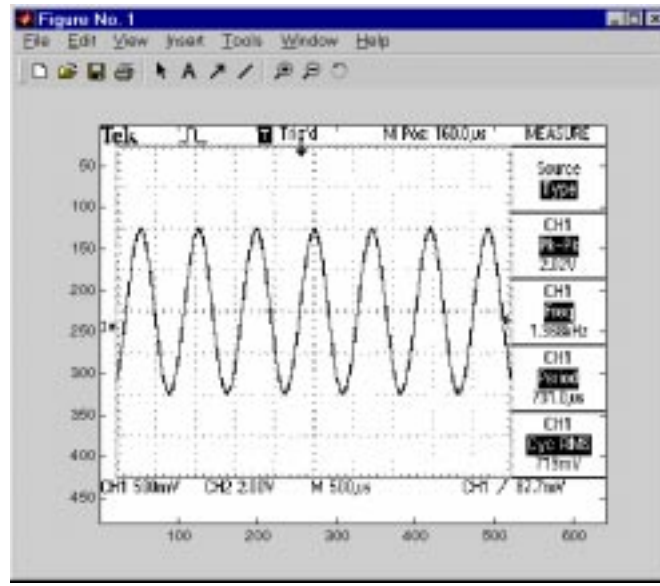
```
fid = fopen('test1.bmp','w');
```

```
fwrite(fid,out,'uint8');
fclose(fid)
a = imread('test1.bmp','bmp');
imagesc(a)
```

スコープは 2 色のみを使ってスクリーン表示を出力するので、適切なカラーマップを選択します。

```
mymap = [0 0 0; 1 1 1];
colormap(mymap)
```

結果のビットマップイメージを、以下に示します。



イベントとアクションの利用

イベント を利用することによって、シリアルポートアプリケーションの能力やフレキシビリティを拡張することができます。イベントは、条件が満たされた後に発生し、1 つまたは複数のアクションを起こします。

シリアルポートオブジェクトがデバイスに接続されているとき、イベントを使ってメッセージを表示し、データを表示し、データを解析し、他のアクションを実行することができます。アクションは、*アクションプロパティ* と *アクション関数* によって制御されます。すべてのイベントタイプは、関連するアクションプロパティを持ちます。アクション関数は、ユーザ独自のアプリケーションの必要に合うように構成した M- ファイル関数です。

M- ファイルアクション関数名を対応するアクションプロパティの値として指定することによって、特定のイベントが発生するときにアクションを実行します。

例題： イベントとアクション

この例題は、M- ファイルアクション関数 instraction を使って、バイトの利用可能なイベントが発生したときにメッセージをコマンドウィンドウに表示します。イベントは、ターミネータが読み込まれたときに生成されます。

```
s = serial('COM1');
fopen(s)
s.BytesAvailableActionMode = 'terminator';
s.BytesAvailableAction = 'instraction';
fprintf(s,'*IDN?')
out = fscanf(s);
```

instraction の結果の表示を以下に示します。

```
BytesAvailable event occurred at 17:01:29 for the object: Serial-COM1.
```

シリアルポートセッションを終了します。

```
fclose(s)
delete(s)
clear s
```

type コマンドを使ってコマンドラインに instraction を表示することができます。

イベントタイプとアクションプロパティ

シリアルポートイベントタイプと対応するアクションプロパティを以下に示します。

表 8-9: イベントタイプとアクションプロパティ

イベントタイプ	対応するプロパティ
Break interrupt	BreakInterruptAction
Bytes available	BytesAvailableAction
	BytesAvailableActionCount
	BytesAvailableActionMode
Error	ErrorAction
Output empty	OutputEmptyAction
Pin status	PinStatusAction
Timer	TimerAction
	TimerPeriod

ブレーク割り込みイベント

ブレーク割り込みイベントは、シリアルポートによるブレーク割り込みが発生した直後に生成されます。シリアルポートは、受信データが 1 キャラクタに対する送信時間よりも長く非アクティブ状態であるときにブレーク割り込みを生成します。

このイベントは、BreakInterruptAction プロパティに対して指定されたアクション関数を実行します。同期および非同期の読み込みおよび書き出し操作に対して生成されます。

Bytes-Available イベント

bytes-available イベントは、あらかじめ決められたバイト数が入力バッファ内で利用可能であるか、BytesAvailableActionMode プロパティで指定されたようにターミネータが読み込まれた直後に生成されます。

BytesAvailableActionMode が byte である場合、bytes-available イベントは、BytesAvailableActionCount で指定されたバイト数が入力バッファに格納されるたびに

BytesAvailableAction プロパティに対して指定されたアクション関数を実行します。BytesAvailableActionModeがterminatorである場合、アクション関数はTerminatorプロパティで指定されたキャラクタが読み込まれるたびに実行します。

このイベントは、非同期読み込み操作中にのみ生成されます。

エラーイベント

エラーイベントは、エラーが発生した直後に生成されます。

このイベントは、ErrorAction プロパティに対して指定されたアクション関数を実行します。非同期の読み込みまたは書き出し操作中にのみ生成されます。

エラーイベントは、タイムアウトが発生したときに生成されます。タイムアウトは、読み込みまたは書き出し操作が Timeout プロパティで指定した時間内に終了しなかった場合に発生します。エラーイベントは、無効なプロパティ値の設定のような設定エラーに対して生成されません。

Output-Empty イベント

output-empty イベントは、出力バッファが空になった直後に生成されます。

このイベントは、OutputEmptyAction プロパティに対して指定されたアクション関数を実行します。非同期の書き出し操作中にのみ生成されます。

Pin Status イベント

pin statusイベントは、CD, CTS, DSR, RIピンに対して状態(ピンの値)が変化した直後に生成されます。これらのピンの説明は、シリアルポートの信号とピンの割り当てを参照してください。

このイベントは、PinStatusAction プロパティに対して指定されたアクション関数を実行します。同期および非同期の読み込みおよび書き出し操作に対して生成されます。

タイマイベント

タイマイベントは、TimerPeriod プロパティで指定した時間が経過したときに生成されます。時間は、シリアルポートオブジェクトがデバイスに接続されたときからの相対時間です。

このイベントは、TimerAction プロパティに対して指定されたアクション関数を実行します。

イベント情報の格納

イベント情報は、2 つのフィールド Type と Data を使って格納されます。Type フィールドはイベントタイプを含み、Data フィールドはイベント固有の情報を含みます。8-52 ページの " アクション関数の作成と実行 " に示すように、これらの2つのフィールドはアクション関数のヘッダで定義する構造体に対応します。データとイベント情報のレコードファイルへの記録に関しては、8-61ページの"デバッグ： 情報のディスクへの記録 " を参照してください。

イベントタイプと Type と Data フィールドに対する値を以下に示します。

表 8-10: イベントの情報

イベントタイプ	フィールド	フィールド値
Break interrupt	Type	BreakInterrupt
	Data.AbsTime	day-month-year hour:minute:second
Bytes available	Type	BytesAvailable
	Data.AbsTime	day-month-year hour:minute:second
Error	Type	Error
	Data.AbsTime	day-month-year hour:minute:second
	Data.Message	An error string
Output empty	Type	OutputEmpty
	Data.AbsTime	day-month-year hour:minute:second
Pin status	Type	PinStatus
	Data.AbsTime	day-month-year hour:minute:second
	Data.Pin	CarrierDetect, ClearToSend, DataSetReady, or RingIndicator
	Data.PinValue	on or off
Timer	Type	Timer
	Data.AbsTime	day-month-year hour:minute:second

Data フィールドの値を以下に示します。

AbsTime フィールド

AbsTime は、すべてのイベントに対して定義され、イベントが発生した絶対時間を示します。絶対時間は、clock フォーマットを使って出力されます。

```
day-month-year hour:minute:second
```

Pin フィールド

Pin は、CD, CTS, DSR, RI ピンが状態を変更したかどうかを示すためにピンステータスイベントによって利用されます。これらのピンに関する説明は、シリアルポート信号とピンの割り当てを参照してください。

PinValue フィールド

PinValue は、CD, CTS, DSR, RI ピンの状態を示すためにピンステータスイベントによって利用されます。利用できる値は、on または off です。

Message フィールド

Messageは、エラーの発生時に生成される記述メッセージを格納するためにエラーイベントによって利用されます。

アクション関数の作成と実行

M- ファイル名を対応するアクションプロパティに対する値として設定することによって、特定のイベントタイプの発生時に実行されるアクション関数を指定することができます。たとえば、デバイスからターミネータが読み込まれたときにアクション関数 myaction を実行するには、

```
s.BytesAvailableActionMode = 'terminator';
s.BytesAvailableAction = 'myaction';
```

M- ファイルアクション関数は、少なくとも 2 つの引数が必要です。第一引数は、シリアルポートオブジェクトです。第二引数は、8-52 ページの表 8-10，イベント情報のイベント情報を格納する変数です。このイベント情報は、アクション関数を実行されるイベントのみに属します。myaction の関数ヘッダを以下に示します。

```
function myaction(obj,event)
```

追加パラメータをセル配列の要素として設定することにより、アクション関数に渡すことができます。たとえば、MATLAB 変数 time を myaction に渡すには、

```
time = datestr(now,0);
s.BytesAvailableActionMode = 'terminator';
```

```
s.BytesAvailableAction = {'myaction',time};
```

対応する関数ヘッダは以下の通りです。

```
function myaction(obj,event,time)
```

追加パラメータをアクション関数に渡す場合は、関数ヘッダの 2 つの必須の引数の後に設定しなければなりません。

エラーの後でアクション関数を利用可能にする

アクション関数の実行中にエラーが発生した場合は、

- アクション関数は自動的に利用不可能になります。
- アクション関数が利用不可能であることを示すワーニングがコマンドラインに表示されます。

同じアクション関数を利用可能にしたい場合は、関数 fclose を使ってオブジェクトを切断しなければなりません。他のアクション関数を利用したい場合は、アクションは、アクションプロパティを新規の値に設定したときに利用可能になります。

例題： イベントとアクションの利用

この例題は、バイト利用可能なイベントまたは出力が空のイベントが発生したときに、コマンドライン上に、イベントに関連する情報を表示するために M- ファイルアクション関数 instraction を利用します。

1．シリアルポートオブジェクトを作成します。 - シリアルポート COM1 に対応するシリアルポートオブジェクト s を作成します。

```
s = serial('COM1');
```

2．デバイスを接続します。 - s を Tektronix TDS 210 オシロスコープに接続します。ReadAsyncMode プロパティに対するデフォルト値は continuous なので、データは機器から利用可能になるとすぐに入力バッファに非同期的に出力されます。

```
fopen(s)
```

3．プロパティを設定します。 - バイト利用可能なイベントあるいは出力が空のイベントの発生時にアクション関数 instraction を行うように s を設定します。

```
s.BytesAvailableActionMode = 'terminator';
s.BytesAvailableAction = 'instraction';
```

```
s.OutputEmptyAction = 'instraction';
```

4．データの書き出しと読み込み - RS232? コマンドをオシロスコープに非同期的に書き出します。このコマンドは、RS-232 設定を質問し、ボーレート、ソフトウェアフローコントロール設定、ハードウェアフローコントロール設定、パリティタイプ、ターミネータを出力します。

```
fprintf(s,'RS232?','async')
```

instraction は、RS232? コマンドが送信された後で、ターミネータが読み込まれたときに呼び出されます。結果の表示を以下に示します。

```
OutputEmpty event occurred at 17:37:21 for the object: Serial-COM1.


BytesAvailable event occurred at 17:37:21 for the object: Serial-COM1.
```

入力バッファからデータを読み込みます。

```
out = fscanf(s)
out =
9600;0;0;NONE;LF
```

5．切断とクリーンアップ - s が必要なくなったときは、機器から切断し、メモリおよび MATLAB ワークスペースから削除します。

```
fclose(s)
delete(s)
clear s
```

制御ピンの使用法

シリアルポート信号とピンの割り当てで説明したように、9 ピンシリアルポートは、6個の制御ピンを含みます。これらの制御ピンを使って以下のことが行えます。

- 接続されているデバイスの存在を知らせる
- データの流れを制御する

シリアルポート制御ピンに対応するプロパティは、以下の通りです。

表 8-11: 制御ピンプロパティ

プロパティ名	説明
DataTerminalReady	DTR ピンの状態を指定します。
FlowControl	利用するデータフロー制御法を指定します。
PinStatus	CD, CTS, DSR, RI ピンの状態を指定をします。
RequestToSend	RTS ピンの状態を指定します。

接続されているデバイスの存在を知らせる

DTE と DCE は、CD, DSR, RI, DTR ピンを使ってシリアルポートとデバイス間で接続が確立されたかどうかを示します。接続が確立されると、データの書き出しや読み込みを開始することができます。

PinStatus プロパティを使って CD, DSR, RI ピンの状態をモニタすることができます。DataTerminalReady プロパティを使って DTR ピンの状態を指定あるいはモニタすることができます。

つぎの例題は、2 つのモデムが互いに接続されているときにこれらのピンがどのように利用されるかを示しています。

例題： 2 つのモデムの接続

この例題は、同じコンピュータから 2 つのモデムを相互に接続し、コンピュータ - モデム、およびモデム - モデム間接続に対する通信の状態をモニタする方法を説明します。1 つ目のモデムは COM1 に接続され、2 つ目のモデムは COM2 に接続されます。

1．シリアルポートオブジェクトを作成します。 - モデムの電源を入れた後で、シリアルポートオブジェクト s1 が第一のモデムに対して作成され、シリアルポートオブジェクト s2 が第二のモデムに対して作成されます。

```
s1 = serial('COM1');
s2 = serial('COM2');
```

2．デバイスに接続します。- s1 と s2 は、モデムに接続されます。ReadAsyncMode プロパティに対するデフォルト値は continuous なので、データは、モデムから利用可能になると直ちに入力バッファに非同期的に出力されます。

```
fopen(s1)
fopen(s2)
```

DataTerminalReady プロパティのデフォルト値は on なので、コンピュータ (データ端末) は、モデムとのデータ交換の準備ができています。PinStatus プロパティを使って Data Set Ready ピンの値を調べることにより、モデム (データセット) はコンピュータとの通信の準備ができていることを確認することができます。

```
s1.Pinstatus
ans =
    CarrierDetect: 'off'
     ClearToSend: 'on'
    DataSetReady: 'on'
    RingIndicator: 'off'
```

両方のモデムはオブジェクトに接続されたときに電源が入っていたので、DataSetReady フィールドの値は on です。

3．プロパティの設定 - 両方のモデムは、ボーレート 2400 ビット / 秒と CR ターミネータに設定されます。

```
s1.BaudRate = 2400;
s1.Terminator = 'CR';
s2.BaudRate = 2400;
s2.Terminator = 'CR';
```

4．データの書き出しと読み込み - atd コマンドを第一のモデムに書き出します。このコマンドは、モデムを "off the hook" にします。これはマニュアルで受話器を離すことと等価です。

```
fprintf(s1,'atd')
```

ata コマンドを第二のモデムに書き出します。このコマンドは、モデムを "answer mode" にします。これは、第一のモデムに接続させます。

```
fprintf(s2,'ata')
```

2 つのモデムがそれらの接続を取り決めた後で、PinStatus プロパティを使って Carrier Detect の値を調べることによって、接続を確認することができます。

```
s1.PinStatus
ans =
    CarrierDetect: 'on'
      ClearToSend: 'on'
     DataSetReady: 'on'
    RingIndicator: 'off'
```

第二のモデムで出力される説明のメッセージを読むことによって、モデム間接続を確認することができます。

```
s2.BytesAvailable
ans =
    25
out = fread(s2,25);
char(out)'
ans =
ata
CONNECT 2400/NONE
```

DataTerminalReady プロパティを off に設定することによって、2 つのモデム間の接続を切断します。Carrier Detect ピンの値を調べることによって、モデムが切断されていることを確認できます。

```
s1.DataTerminalReady = 'off';
s1.PinStatus
ans =
    CarrierDetect: 'off'
      ClearToSend: 'on'
     DataSetReady: 'on'
    RingIndicator: 'off'
```

5．**切断とクリーンアップ** - オブジェクトをモデムから切断し、メモリおよびMATLAB ワークスペースからオブジェクトを削除します。

```
fclose([s1 s2])
delete([s1 s2])
clear s1 s2
```

データフローの制御： ハンドシェイク

データフローコントロールまたは*ハンドシェイク*は、送信中のデータのロスを防ぐために DCE と DTE 間の通信に用いられる方法です。たとえば、コンピュータが処理される前に限られた量のデータしか受信しないと仮定します。この制限に達すると、データの送信を中止するためにハンドシェイク信号が DCE に送信されます。コンピュータがより多くのデータを受信できるときは、他のハンドシェイク信号が DCE に送信されてデータの送信を再開します。

デバイスがサポートしている場合は、つぎの方法のいずれかを使ってデータフローを制御することができます。

- Hardware handshaking
- Software handshaking

注意　ハードウェアハンドシェイクとソフトウェアハンドシェイクの両方をデバイスに対して同時に設定することは可能ですが、MATLAB はこの挙動をサポートしません。

FlowControl プロパティを使ってデータフロー制御方法を指定できます。FlowControl が hardware である場合は、データフローの制御にハードウェアハンドシェイクが利用されます。FlowControl が software である場合は、データフローの制御にソフトウェアハンドシェイクが利用されます。FlowControl が none，である場合は、ハンドシェイクは用いられません。

ハードウェアハンドシェイク

ハードウェアハンドシェイクは、特定のシリアルポートピンを使ってデータフローを制御します。ほとんとの場合で、これらは RTS ピンと CTS ピンです。これらのピンを使ったハードウェアハンドシェイクは、8-10 ページの "RTS ピンと CTS ピン ” に説明されています。

FlowControlがhardwareである場合は、RTSピンとCTSピンは、DTEとDCEによって自動的に管理されます。PinStatus プロパティを使って CTS ピンの値を出力することができます。RequestToSend プロパティを使って RTS ピンの値を設定あるいは出力することができます。

注意　デバイスの中には、ハンドシェイク用に DTR ピンと DSR ピンを利用するものがあります。しかし、これらのピンは、主として通信用にシステムの準備が完了していることを示すために利用され、データ送信を制御するためには利用されません。MATLAB では、ハードウェアハンドシェイクは常に RTS ピンと CTS ピンを使います。

デバイスが標準の方法でハードウェアハンドシェイクを利用しない場合は、RequestToSend プロパティをマニュアルで設定する必要があります。この場合、FlowControl を none に設定します。FlowControl が hardware である場合は、指定する RequestToSend 値は与えられません。ピンの挙動を指定するためには、デバイスのドキュメントを参照してください。

ソフトウェアハンドシェイク

ソフトウェアハンドシェイクは、特定の ASCII キャラクタを使ってデータフローを制御します。Xon および Xoff(または XON および XOFF) として知られているこれらのキャラクタを、以下に説明します。

表 8-12: ソフトウェアハンドシェイクキャラクタ

キャラクタ	値	説明
Xon	17	データ送信を再開
Xoff	19	データ送信を中止

ソフトウェアハンドシェイクの利用時に、コントロールキャラクタは、通常のデータと同じ方法で送信ライン上で送信されます。そのため、TD, RD, GND ピンのみが必要です。

ソフトウェアハンドシェイクの主な短所は、数値データがデバイスに書き出されているときに Xon あるいは Xoff キャラクタを書き出せないことです。これは、数値データは 17 または 19 を含む場合があり、コントロールキャラクタとデータを区別できないためです。しかし、データをデバイスから非同期的に読み込んでい

るときには TD と RD ピンを使っているので、Xon または Xoff を書き出すことができます。

例題： ソフトウェアハンドシェイクの利用

8-45 ページの "例題 : バイナリデータの読み込み " で説明した例題を使ってソフトウェアフロー制御を利用したいと仮定します。そのためには、ソフトウェアフロー制御用にオシロスコープとシリアルポートオブジェクトを設定する必要があります。

```
fprintf(s,'RS232:SOFTF ON')
s.FlowControl = 'software';
```

データ送信を中止するには、数値 19 をデバイスに書きます。

```
fwrite(s,19)
```

データ送信を再開するには、数値 17 をデバイスに書きます。

```
fwrite(s,17)
```

デバッグ： 情報をディスクに記録

シリアルポートオブジェクトがデバイスに接続されているときは、つぎの情報をディスクファイルに記録することができます。

- デバイスに書き出される値の数、デバイスから読み込まれる値の数、値のデータタイプ。
- デバイスに書き出されるデータ、デバイスから読み込まれるデータ。
- イベント情報

情報のディスクへの保存は、シリアルポートセッションの不変レコードを与え、アプリケーションのデバッグのための簡単な方法です。

関数 record を使ってディスクファイルに情報を記録します。ディスクへの情報の記録に関連するプロパティを以下に示します。

表 8-13: Recording プロパティ

プロパティ名	説明
RecordDetail	レコードファイルに保存される情報量を指定します。
RecordMode	データとイベント情報が1つのレコードファイルあるいは複数のレコードファイルに保存されるかを指定します。
RecordName	レコードファイル名を指定します。
RecordStatus	データとイベント情報がレコードファイルに保存されるかどうかを指定します。

例題： 情報の記録

この例題は、デバイスから書き出されたり読み込まれる値の数を記録し、ファイル myfile.txt に情報を保存します。

```
s = serial('COM1');
fopen(s)
s.RecordName = 'myfile.txt';
record(s)
fprintf(s,'*IDN?')
idn = fscanf(s);
```

```
fprintf(s,'RS232?')
rs232 = fscanf(s);
```

シリアルポートセッションを終了します。

```
fclose(s)
delete(s)
clear s
```

type コマンドを使って、コマンドラインに myfile.txt を表示することができます。

複数のレコードファイルの作成

関数 record を使って記録を開始するとき、RecordMode プロパティは、新規のレコードファイルが作成されるか、あるいは新規の情報が既存のレコードファイルに付加されるかを決定します。

RecordModeをoverwrite, append, indexに設定できます。RecordModeがoverwriteの場合、レコードファイルは記録が開始されるたびに上書きされます。RecordMode が append の場合は、新規の情報が RecordName で指定されたファイルに追加されます。RecordMode が index の場合は、記録が開始されるたびに別のディスクファイルが作成されます。レコードファイル名の指定の規則は、つぎの節で説明します。

ファイル名の指定

RecordName プロパティを使ってレコードファイル名を指定します。ファイル名がオペレーティングシステムによってサポートされている限り、ディレクトリパスを含む任意の値を RecordName に対して指定することができます。さらに、RecordMode が index の場合は、ファイル名は、次の規則に従います。

- インデックス付きのファイル名は、数字によって識別されます。この数字は、ファイル名の拡張子の前にあり、連続するレコードファイルに対して 1 ずつ増加します。
- 初期のファイル名の一部として数字が指定されない場合は、最初のレコードファイルはその数字に対応する数字を持ちません。たとえば、RecordName が myfile.txt の場合は、myfile.txt は最初のレコードファイル名で、myfile01.txt が 2 番目のレコードファイル名である、等です。
- RecordName は、レコードファイルがクローズされた後に更新されます。
- 指定したファイル名が既に存在する場合は、既存のファイルが上書きされます。

レコードファイルフォーマット

レコードファイルは、1 つあるいは複数のシリアルポートセッションの記録を含む ASCII ファイルです。RecordDetail プロパティを使ってレコードファイルに保存される情報量を指定します。

RecordDetail は、compact あるいは verbose です。compact レコードファイルは、デバイスに書き出される値の数、デバイスから読み込まれる値の数、値のデータタイプ、イベント情報を含みます。verbose レコードファイルは、デバイスから送信されたデータとその前の情報を含みます。

uchar, schar, (u)int8, (u)int16, (u)int32 で与えられた精度を持つバイナリデータは、16 進数フォーマットを使って記録されます。たとえば、10 進数 255 が 16 ビット整数として機器から読み込まれた場合は、16 進数 00FF がレコードファイルに保存されます。単精度および倍精度数は、IEEE Standard 754-1985 for Binary Floating-Point Arithmetic で指定されたフォーマットを使って記録されます。

IEEE 浮動小数点フォーマットは、符号ビット、指数フィールド、仮数フィールドの 3 つのコンポーネントを持ちます。単精度浮動小数点数は、32 個のビットで構成され、値はつぎのように与えられます。

$$\text{value} = (-1)^{\text{sign}}(2^{\text{exp-127}})(1.\text{significand})$$

倍精度浮動小数点数は、64 個のビットで構成され、値はつぎのように与えられます。

$$\text{value} = (-1)^{\text{sign}}(2^{\text{exp-1023}})(1.\text{significand})$$

浮動小数点フォーマットコンポーネントと対応する単精度および倍精度ビットは、以下の通りです。

コンポーネント	単精度ビット	倍精度ビット
sign	1	1
exp	2-9	2-12
significand	10-32	13-64

ビット 1 は、レコードファイルに保存される最も左のビットです。

単精度および倍精度数は、%g フォーマットを使って右側に表示された値のテキスト表現も行います。

例題： 情報をディスクに記録

この例題は、シリアルポートオブジェクトと Tektronix TDS 210 オシロスコープ間で送信される情報を記録する方法を説明します。さらに、結果のレコードファイルの構造も説明します。

1．シリアルポートオブジェクトを作成します。 - シリアルポート COM1 に対応するシリアルポートオブジェクト s を作成します。

```
s = serial('COM1');
```

2．デバイスを接続します。 - s をオシロスコープに接続します。ReadAsyncMode プロパティに対するデフォルト値は continuous なので、データは機器から利用可能になるとすぐに入力バッファに非同期的に出力されます。

```
fopen(s)
```

3．プロパティ値の設定 - verbose フォーマットを使って s を複数のディスクファイルに情報を記録するように設定します。記録は、WaveForm1.txt として定義された最初のディスクファイルから始まります。

```
s.RecordMode = 'index';
s.RecordDetail = 'verbose';
s.RecordName = 'WaveForm1.txt';
record(s)
```

4．データの書き出しと読み込み - 機器に書き出されるコマンドと機器から読み込まれるデータは、レコードファイルに記録されます。オシロスコープのコマンドの説明は、例題： テキストデータの書き出しと読み込みを参照してください

```
fprintf(s,'*IDN?')
idn = fscanf(s);
fprintf(s,'MEASUREMENT:IMMED:SOURCE CH2')
fprintf(s,'MEASUREMENT:IMMED:SOURCE?')
source = fscanf(s);
```

関数 fread を使ってピーク間電圧を読み込みます。fread から出力されるデータは、hex フォーマットを使って記録されます。

```
fprintf(s,'MEASUREMENT:MEAS1:TYPE PK2PK')
fprintf(s,'MEASUREMENT:MEAS1:VALUE?')
ptop = fread(s,s.BytesAvailable);
```

ピーク間電圧をキャラクタ配列に変換します。

```
char(ptop)'
ans =
2.0199999809E0
```

記録の状態は、on から off に切り替えられます。RecordMode の値は index なので、レコードファイル名は自動的に更新されます。

```
record(s)
s.RecordStatus
ans =
off
s.RecordName
ans =
WaveForm2.txt
```

5．切断とクリーンアップ - s が必要なくなったときは、機器から切断し、メモリおよび MATLAB ワークスペースから削除します。

```
fclose(s)
delete(s)
clear s
```

レコードファイルの内容

WaveForm1.txt レコードファイルの内容を、以下に示します。RecordDetail プロパティは verbose なので、値の数、コマンド、データが記録されています。関数 fread から出力されたデータは、hex フォーマットです。

```
type WaveForm1.txt

Legend:
 * - An event occurred.
 > - A write operation occurred.
 < - A read operation occurred.

1    Recording on 22-Jan-2000 at 11:21:21.575. Binary data in...
2   > 6 ascii values.
     *IDN?
3   < 56 ascii values.
     TEKTRONIX,TDS 210,0,CF:91.1CT FV:v1.16 TDS2CM:CMV:v1.04
4   > 29 ascii values.
     MEASUREMENT:IMMED:SOURCE CH2
5   > 26 ascii values.
     MEASUREMENT:IMMED:SOURCE?
6   < 4 ascii values.
     CH2
7   > 27 ascii values.
     MEASUREMENT:MEAS1:TYPE PK2PK
8   > 25 ascii values.
     MEASUREMENT:MEAS1:VALUE?
9   < 15 uchar values.
     32 2e 30 31 39 39 39 39 39 38 30 39 45 30 0a
10   Recording off.
```

保存とロード

シリアルポートオブジェクトは、save コマンドを使ってワークスペース変数のように MAT- ファイルに保存することができます。たとえば、シリアルポート COM1 に対応するシリアルポートオブジェクト s を作成し、プロパティ値を設定し、書き出しおよび読み込み操作を行うと仮定します。

```
s = serial('COM1');
s.BaudRate = 19200;
s.Tag = 'My serial object';
fopen(s)
fprintf(s, '*IDN?')
out = fscanf(s);
```

シリアルポートオブジェクトとデバイスから読み込んだデータを MAT- ファイル myserial.mat に保存するには、つぎのようにします。

```
save myserial s out
```

注意　関数 record を使って、データとイベント情報をテキストとしてディスクファイルに保存することができます。

---

load コマンドを使って、s と out をワークスペースに再生することができます。

```
load myserial
```

参照のみのプロパティの値は、ロード時にデフォルト値にリストアされます。たとえば、Status プロパティは、closed にリストアされます。そのため、s を利用するには、関数 fopen を使ってデバイスに接続しなければなりません。プロパティが参照のみかどうかを決定するには、リファレンスページを調べてください。

異なるプラットフォームでシリアルポートオブジェクトを利用

あるプラットフォームでシリアルポートオブジェクトを保存し、その後オブジェクトを異なるシリアルポート名をもつ別のプラットフォームでロードする場合、Port プロパティ値を変更する必要があります。たとえば、Windows プラットフォームでシリアルポート COM1 に対応するシリアルポートオブジェクト s を作成すると仮定します。最終的に Linux プラットフォームで利用するために s を保存したい場合は、オブジェクトをロードした後で、ttyS0 のような適切な値を Port に設定します。

切断とクリーンアップ

シリアルポートオブジェクトが必要でなくなったときには、デバイスから切断し、メモリおよびワークスペースからオブジェクトを削除することによって、MATLAB 環境をクリーンアップします。これは、シリアルポートセッションを終了するために行うべきことです。

シリアルポートオブジェクトの切断

デバイスと通信を行う必要がなくなったときには、関数 fclose を使ってシリアルポートオブジェクトから切断します。

```
fclose(s)
```

シリアルポートオブジェクトとデバイスが切断されていることを確認するために Status プロパティを調べることができます。

```
s.Status
ans =
closed
```

fclose を実行した後で、s に対応するシリアルポートは利用可能です。fopen を使って他のシリアルポートオブジェクトに接続することができます。

MATLAB 環境のクリーンアップ

シリアルポートオブジェクトが必要でなくなったときには、関数 delete を使ってメモリから削除します。

```
delete(s)
```

delete を使う前に、関数 fclose を使って、シリアルポートオブジェクトをデバイスから切断しなければなりません。

削除されたシリアルポートオブジェクトは*無効*で、これは、デバイスに接続できないことを意味します。この場合には、MATLAB ワークスペースからオブジェクトを削除します。シリアルポートオブジェクトと他の変数を MATLAB ワークスペースから削除するには、clear コマンドを使います。

```
clear s
```

まだデバイスに接続されているシリアルポートオブジェクトに対してclearを使う場合は、オブジェクトはワークスペースから削除されますが、デバイスには接続

されています。関数 instrfind を使ってクリアされたオブジェクトを MATLAB にリストアできます。

シリアルポート関数

シリアルポートオブジェクト関数が、つぎに簡単に記述され、使用法をベースに、カテゴリィ別にまとめられています。より詳しい記述は、ヘルプブラウザの中の "MATLAB 関数リファレンス“ を参照してください。

シリアルポートオブジェクトの作成	
serial	シリアルポートオブジェクトの作成

データの書き出しと読み取り	
fgetl	デバイスからテキストを 1 行、読み取り、終端子を削除
fgets	デバイスからテキストを 1 行、読み取り、終端子も含む
fprintf	テキストをデバイスに書き出す
fread	デバイスからバイナリデータを読み取る
fscanf	デバイスからデータを読み取り、テキストとしてフォーマット
fwrite	バイナリデータをデバイスに書き出す
readasync	デバイスから非同期状態でデータを読み取る
stopasync	非同期状態の読み取りと書き出しを停止

プロパティの取得と設定	
get	シリアルポートオブジェクトを出力
set	シリアルポートオブジェクトの設定、または、表示

状態の変更	
fclose	デバイスとシリアルポートオブジェクトを切断
fopen	デバイスとシリアルポートオブジェクトを接続
record	データとイベント情報をファイルに記録

その他一般	
clear	MATLAB ワークスペースからシリアルポートオブジェクトを削除
delete	メモリからシリアルポートオブジェクトを削除
disp	シリアルポートオブジェクトの情報を表示
instraction	イベントが生じたとき、イベント情報を表示
instrfind	メモリから MATLAB ワークスペースにシリアルポートオブジェクトを出力
isvalid	シリアルポートオブジェクトが正しいか否かの検索
length	シリアルポートオブジェクト配列の長さ
load	シリアルポートオブジェクトや変数を MATLAB ワークスペースにロード
save	MAT ファイルにシリアルポートオブジェクトや変数をセーブ
serialbreak	シリアルポートに接続しているデバイスを切断
size	シリアルポートオブジェクト配列のサイズ

いくつかの他の M- ファイル補助関数が、上の表に記述されている関数をサポートするために、シリアルポートオブジェクトと共に用意されています。これらの補助関数のドキュメントは、ユーザが直接使用しないのでありません。

Serial Port プロパティ

" シリアルポートオブジェクトプロパティ " は、利用法に基づくカテゴリを使ってプロパティをまとめています。より詳細な記述は、ヘルプブラウザの中の "External Interfaces/API" を参照してください。

Communications プロパティ	
BaudRate	ビットが送信されるレートを指定します。
DataBits	送信されるデータビット数を指定します。
Parity	パリティチェックのタイプを指定します。
StopBits	バイトの終了を示すビット数を指定します。
Terminator	ターミネータキャラクタを指定します。

Write プロパティ	
BytesToOutput	現在出力バッファにあるバイト数を示します。
OutputBufferSize	出力バッファのサイズをバイト単位で指定します。
Timeout	読み込みまたは書き出し操作を終了するまでの待ち時間を指定します。
TransferStatus	進行中の非同期読み込みまたは書き出し操作を示します。
ValuesSent	デバイスに書き出される値の総数を示します。

Read プロパティ	
BytesAvailable	入力バッファに格納可能なバイト数を示します。
InputBufferSize	入力バッファのサイズをバイト単位で指定します。

Read プロパティ	
ReadAsyncMode	非同期読み込み操作が連続であるかマニュアルであるかを指定します。
Timeout	読み込みまたは書き出し操作を終了するまでの待ち時間を指定します。
TransferStatus	進行中の非同期読み込みまたは書き出し操作を示します。
ValuesReceived	デバイスから読み込まれる値の総数を示します。

Action プロパティ	
BreakInterrupt Action	ブレーク割り込みイベントの発生時に実行する M- ファイルアクション関数を指定します。
BytesAvailable Action	指定したバイト数が入力バッファで利用可能か、あるいはターミネータが読み込まれたときに実行する M- ファイルアクション関数を指定します。
BytesAvailable ActionCount	バイト利用可能なイベントを生成するために入力バッファで利用可能なバイト数を指定します。
BytesAvailable ActionMode	指定したバイト数が入力バッファで利用可能か、あるいはターミネータが読み込まれた後でバイト利用可能なイベントが生成されるかどうかを指定します。
ErrorAction	エラーイベントの発生時に実行する M- ファイルアクション関数を指定します。
OutputEmpty Action	出力バッファが空のときに実行する M- ファイルアクション関数を指定します。
PinStatus Action	CD, CTS, DSR, RI ピンが状態を変更したときに実行する M- ファイルアクション関数を指定します。

Action プロパティ	
TimerAction	あらかじめ定義された時間が経過したときに実行する M- ファイルアクション関数を指定します。
TimerPeriod	タイマイベント間の時間を指定します。

Control Pin プロパティ	
DataTerminal Ready	DTR ピンの状態を指定します。
FlowControl	利用するデータフロー制御法を指定します。
PinStatus	CD, CTS, DSR, RI ピンの状態を示します。
RequestToSend	RTS ピンの状態を指定します。

Recording プロパティ	
RecordDetail	レコードファイルに保存される情報量を指定します。
RecordMode	データとイベント情報が 1 つのレコードファイルに保存されるか、あるいは複数のレコードファイルに保存されるかを指定します。
RecordName	レコードファイル名を指定します。
RecordStatus	データとイベント情報がレコードファイルに保存されるかどうかを示します。

General Purpose プロパティ	
ByteOrder	デバイスがバイトを格納する順番を指定します。
Name	シリアルポートオブジェクトを説明する名前を指定します。
Port	プラットフォーム固有のシリアルポート名を指定します。
Status	シリアルポートオブジェクトがデバイスに接続されているかどうかを示します。
Tag	シリアルポートオブジェクトに対応するラベルを指定します。
Type	オブジェクトタイプを指定します。
UserData	シリアルポートオブジェクトと対応付けたいデータを指定します。

参考文献

1 Axelson, Jan, *Serial Port Complete*, Lakeview Research, Madison, WI, 1998.

2 *Courier High Speed Modems User's Manua*l, U.S. Robotics, Inc., Skokie, IL, 1994.

3 *Getting Started with Your AT Serial Hardware and Software for Windows 98/95*, National Instruments, Inc., Austin, TX, 1998.

4 *Instrument Communication Handbook*, IOTech, Inc., Cleveland, OH, 1991.

5 *TDS 200-Series Two Channel Digital Oscilloscope Programmer Manual*, Tektronix, Inc., Wilsonville, OR.

6 TIA/EIA-232-F, *Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange*.

お問い合わせ先

サイバネットシステム株式会社
本社　東京都文京区大塚 2 丁目 15 番地 6 号
ニッセイ音羽ビル

応用システム第一営業部
電話　03-5978-5410
FAX　03-5978-5440
応用システム第一技術部
電話　03-5978-5411
FAX　03-5978-5440

大阪支社
大阪市中央区常盤町 1-3-8
中央大通 FN ビル 20 階
電話　06-6940-3610
FAX　06-6940-3602