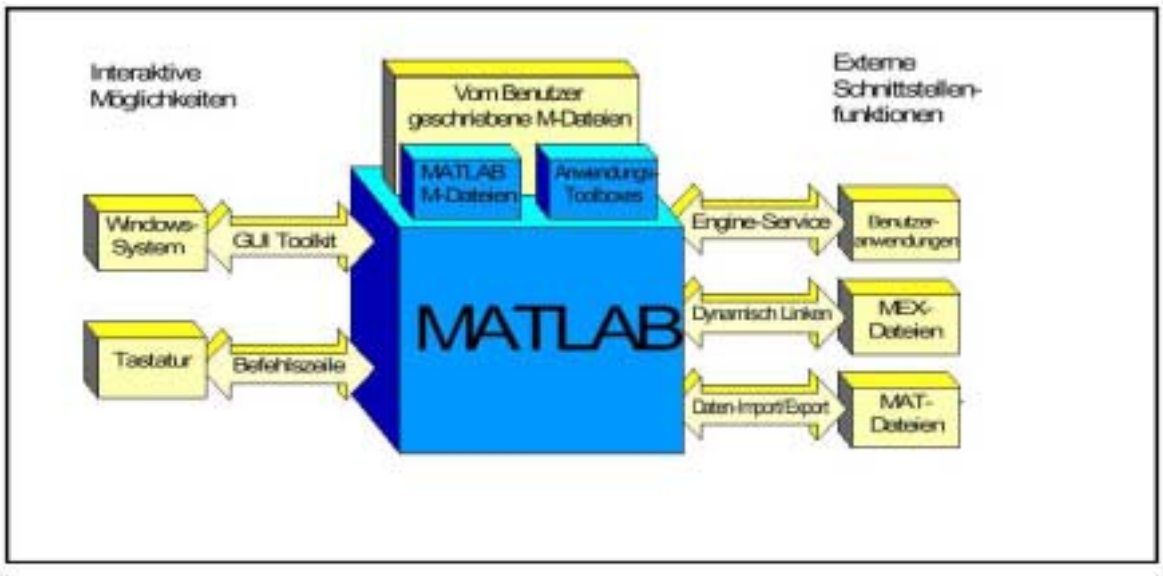
MATLAB – MEX - Ein offenes System



Über die *MEX*[1]*-Schnittstelle* können eigene *C/C++*, *Fortran* und *Java* Programme in den offenen Softwareframework *MATLAB* zur Laufzeit eingebunden werden. *MATLAB Mex-Programme* verhalten sich nach außen wie alle anderen standardmäßig eingebauten *MATLAB-Funktionen*. *MEX-Programme* in der Programmiersprache C werden in einer .c-Datei mit dem Namen der Funktion gespeichert. Beispiel: die Funktion *test* wird in der Datei *test.c* implementiert. In der .c-Datei muss eine Funktion *mexFunction* mit folgender Schnittstelle definiert sein:

```
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
```

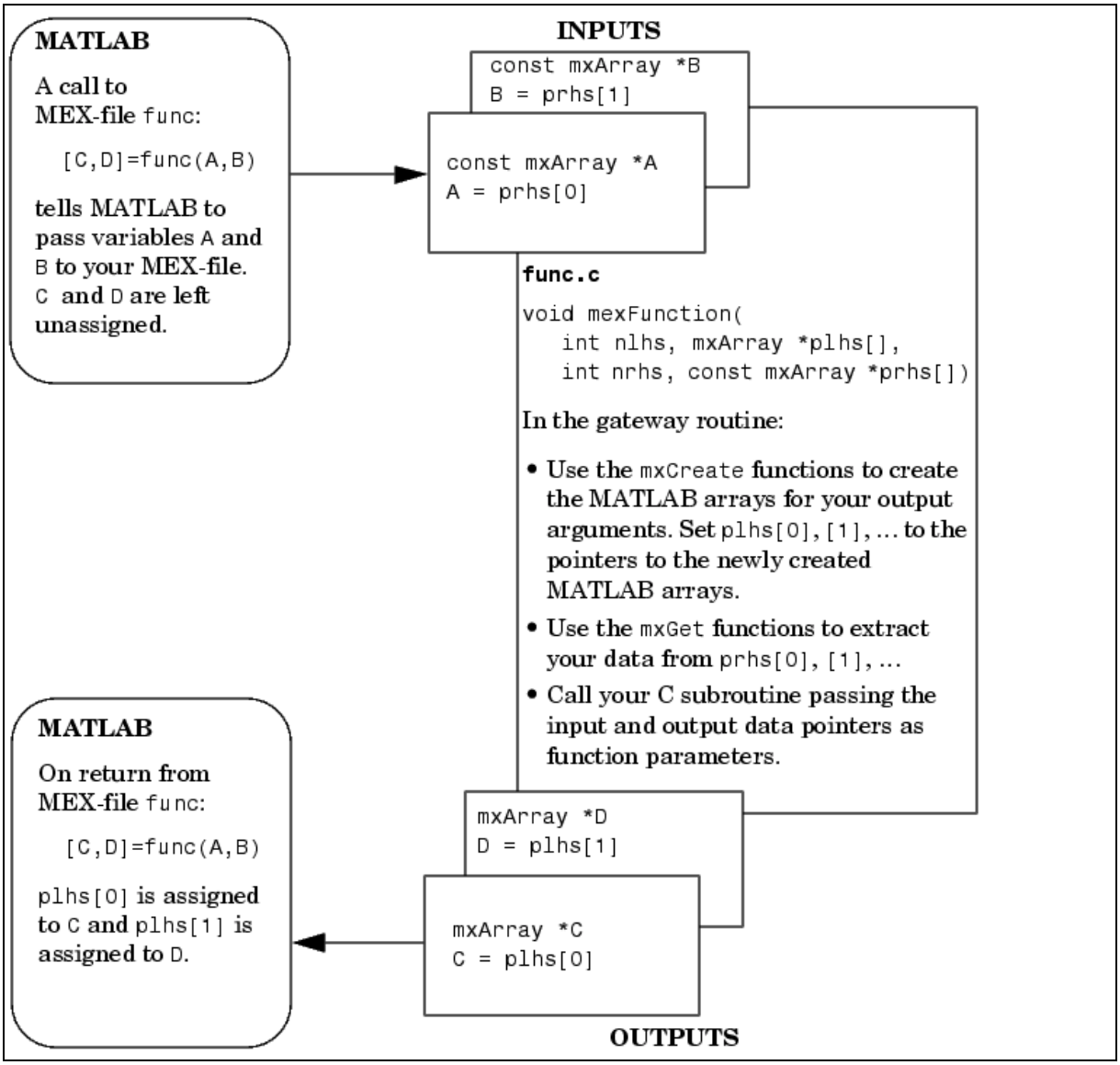
Dieser Funktion werden vier Parameter übergeben:

1. ***int nlhs:*** Anzahl der Rückgabeparameter (linke Seite)
2. ***mxArray *plhs[]:*** ein Feld von Pointern auf die Rückgabeparameter (mxArray). Dabei ist plhs[0] der erste Parameter, plhs[1] der zweite Parameter, usw. Die Rückgabeparameter sind undefiniert und müssen in der Funktion *mexFunction* erzeugt werden.
3. ***int nrhs:*** Anzahl der Eingabeparameter (rechte Seite).
4. ***const mxArray *prhs[]:*** ein Feld von Pointern auf die Eingabevariablen (mxArray).

[1] MEX = MATLAB External Interface

Für den Zugriff auf Daten vom Typ *mxArray* stehen dem Programmierer die Funktionen aus der Bibliothek *libmx.lib*, *matrix.h* zur Verfügung. Weitere nützliche Funktionen u.a. für die Ausgabe von (Fehler-)Meldungen befinden sich in der Bibliothek *libmex.lib*, *mex.h*:

```
#include "matrix.h"

...

/*
 * Issue error message and return to MATLAB prompt
 */
extern void mexErrMsgTxt(
    const char  *error_msg /* string with error message */
    );

/*
 * mex equivalent to MATLAB's "disp" function
 */
extern int mexPrintf(
    const char  *fmt,      /* printf style format */
    ...                    /* any additional arguments */
    );

#define printf mexPrintf

/*
 * mexFunction is the user defined C routine which is called
 * upon invocation of a mex function.
 */
void mexFunction(
  int            nlhs,   /* number of expected outputs */
  mxArray       *plhs[], /* array of pointers to output args */
  int            nrhs,   /* number of inputs */
  const mxArray *prhs[]  /* array of pointers to input args */
);
```


MATLAB
A call to MEX-file func:
[C,D]=func(A,B)
tells MATLAB to pass variables A and B to your MEX-file. C and D are left unassigned.
INPUTS
const mxArray *B
B = prhs[1]
const mxArray *A
A = prhs[0]
func.c
void mexFunction(
int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[])
In the gateway routine:
• Use the mxCreate functions to create the MATLAB arrays for your output arguments. Set plhs[0], [1], ... to the pointers to the newly created MATLAB arrays.
• Use the mxGet functions to extract your data from prhs[0], [1], ...
• Call your C subroutine passing the input and output data pointers as function parameters.
mxArray *D
D = plhs[1]
mxArray *C
C = plhs[0]
OUTPUTS
MATLAB
On return from MEX-file func:
[C,D]=func(A,B)
plhs[0] is assigned to C and plhs[1] is assigned to D.

Einführendes Beispiel

Als Beispiel diene die *MEX-Funktion first*. Hier soll eine Matrix erstellt werden, in der jede Zeile mit der jeweiligen Zeilennummer gefüllt wird:

```
>> a = first(3,4)

a =
   1   1   1   1
   2   2   2   2
   3   3   3   3
```

```
first(2,5)

ans =
   1   1   1   1   1
   2   2   2   2   2
```

Die Funktion hat zwei Eingabeparameter. Diese legen die Größe der Matrix fest (Zeilen, Spalten). Ergebnis ist die erzeugte Matrix.

Die Funktion wird programmiert in der Datei *first.c*. Mit Hilfe des MATLAB-Kommandos *mex* wird mit Hilfe des C-Compilers und Linkers eine dynamische *DLL* generiert:

```
>> mex –v first.c
```

Sind auf dem Rechner mehrere Compiler installiert, kann durch den Aufruf von *mex –setup* einer eingestellt werden:

```
>> mex -setup
Please choose your compiler for building external
interface (MEX) files:

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

Select a compiler:
[1] Lcc C version 2.4 in C:\MATLAB6P5\sys\lcc
[2] Microsoft Visual C/C++ version 6.0

[0] None

Compiler: 2
```

```c
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    double *pWerte;
    int zeilen, spalten, z, s;

    /* Prüfe Anzahl der Eingabeparameter */
    if ( nrhs != 2 || nlhs > 1)
        mexErrMsgTxt("Falscher Aufruf");

    /* Erster Eingabeparameter muss Skalar sein */
    if ( ! (mxGetM(prhs[0]) == 1 && mxGetN(prhs[0]) == 1) )
        mexErrMsgTxt("Erster Parameter muss Skalar sein");

    /* Zweiter Eingabeparameter muss Skalar sein */
    if (! (mxGetM(prhs[1]) == 1 && mxGetN(prhs[1]) == 1) )
        mexErrMsgTxt("Zweiter Parameter muss Skalar sein");

    /* Parameter zeilen und spalten auslesen */
    zeilen  = (int) mxGetScalar(prhs[0]);
    spalten = (int) mxGetScalar(prhs[1]);

    /* Ergebnismatrix erstellen */
    plhs[0] = mxCreateDoubleMatrix(zeilen, spalten, mxREAL);
    pWerte = mxGetPr(plhs[0]);

    /* Werte in Matrix füllen, Spaltenweise dicht */
    for (z=0; z<zeilen; z++)
        for (s=0; s<spalten; s++)
            pWerte[s*zeilen+z] = z+1;
}
```

Unterschiede C / C++

	C	**C++**
	Prozedurale Sprache (Funktionen)	= C + OOP (*class*)
Vorteile	Effizienz (Zeit, Speicher)	Wiederverwendbarkeit Klassenbibliotken (*MFC*) Templates (*STL*)
Anwendung	Hardwarenahe Programmierung, Signalprozessoren, DSP	Zeitunkritische Anwendungen, Grafische Programme
Parameterübergabe bei Funktionsaufrufen	*Call by value*	*Call by value* und *Call by reference* (int &)
Dynamische Programmierung	Funktionen *malloc* und *free* aus *malloc.h*	Operatoren ***new*** und ***delete***
Ein- und Ausgabe	Funktionen *printf* und *scanf* aus *stdio.h*	Klassen *cin* und *cout* aus *iostream.h*
Datei	Funktionen *fopen*, *fclose* *fprintf*, *fscanf* für ASCII-Dateien *fwrite*, *fread* für Binärdateien	Klassen *ifstream*, *ofstream* aus *fstream.h*
Ausnahmebehandlung	Funktionen aus *signal.h*	Exceptions – *try*, *catch*
Sonstiges	Vor erster Anweisung müssen alle verwendeten Variablen bzw Funktionen deklariert sein!	*Templates* für Datentyp-unabhängige Programmierung *Overloading*: gleicher Methodenname, unterschiedliche Parameterliste *Defaultwerte* für Methodenparameter
Nützliche C Funktionen	process.h – Process Control memory.h - RAM direct.h: Directory Control floats.h: Genauigkeit string.h: Zeichenketten	

Für den Aufruf von *C*-Funktionen aus *C++* müssen die *C*-Funktionen mit der sogenannten *C-Bindung* deklariert sein. Dies geschieht in der Regel in der *Headerdatei* der C-Funktion.

Beispiel *sort.h:*

```
#ifndef _SORT_
#define _SORT_

#ifdef __cplusplus
extern "C" {
#endif

void BubbleSort(double feld[], int anzahl, int sortflag);


#ifdef __cplusplus
}
#endif

#endif
```

Über das *__cplusplus* Makro wird festgestellt, ob C oder C++ kompiliert wird. Dieses Makro ist nur beim C++ Compiler gesetzt.