

---

## ■ Inhaltsverzeichnis

1.1	Korrespondierende Typen . . . . .	1
1.2	Symbolnamen. . . . .	2
1.3	Der Stack - Argumentenübergabe und Freigabe . . . . .	2
1.4	Runtime Libraries . . . . .	4
1.5	Zusammenfassung . . . . .	4
2.	Fortran wird gerufen ("C calls Fortran") . . . . .	5
3.	Fortran ruft ("Fortran calls C") . . . . .	8
3.1	Schnittstellendeklaration mittels INTERFACE . . . . .	8
3.1.1	VF Compiler Direktiven - "DEC\$ ATTRIBUTES ...". . . . .	8
3.1.2	Fortran 2003 Befehle BIND und VALUE. . . . .	10
3.2	Sonderbehandlung von Strings. . . . .	10
4.	Inkompatibilitäten . . . . .	11
5.	Beispiele / Projekte: . . . . .	13

## 5. Beispiele / Projekte:

Die Visual Studio Solution `MixedLanguageProgramming.sln` enthält diverse IVF und VC Projekte (`.vcproj`, `.vfproj`), die beispielhaft das zuvor Gesagte illustrieren sollen.

A. `FLibStatic` (f90):  
erstellt eine statische Bibliothek (`.lib`) aus diversen Fortran SUBROUTINES, FUNCTIONS und MODULES, die von den anderen Projekten verwendet wird.

B. `FLibDynamic` (f90):  
erstellt eine dynamische Bibliothek (`.dll`) aus den gleichen Fortran SUBROUTINES,

FUNCTIONS und MODULES wie `FLibStatic`. Man beachte die Exportdatei `DLLExports.def`, die für die "Publizierung" der Symbole sorgt. Die `FLibDynamic` wird von anderen Projekten verwendet.

1. `Ex01_Main` (f90):

`FMain` ruft die SUBROUTINES `FSub01`, `FSub04` und die FUNCTION `FRFun01` aus der `FLibStatic`. Das Programm dient der "Einstimmung" und zum Nachweis, daß die Fortran Routinen, die auch in den C++ Programmen verwendet werden, unverändert innerhalb eines Fortran Programms gerufen werden können.

2. `Ex02_Main` (f90):

`FMain` ruft die SUBROUTINES `FSub01`, `FSub04` und die FUNCTION `FRFun01` aus der `FLibDynamic`. Es dient zum einen dem gleichen Zweck wie `Ex01_Main` und zeigt überdies die Verwendung einer DLL.

3. `Ex01_C_calls_Fortran` (cpp):

Das C++ Programm ruft die Fortran SUBROUTINE `FSub01( i2Arg, r4Arg, i4Arg, r8Arg )` aus der `FLibStatic`. Es zeigt Grundsätzliches zum Aufruf einer Fortran SUBROUTINE und zur Typkompatibilität zwischen Fortran und C/C++.

4. `Ex02_C_calls_Fortran` (cpp):

Das C++ Programm ruft die SUBROUTINE `FSub01( i2Arg, r4Arg, i4Arg, r8Arg )` und die FUNCTION `FRFun01( rArg, sText1, iArg, sText2 )` `RESULT(lRet)`

aus der `FLibDynamic`. Es zeigt den Aufruf einer Fortran SUBROUTINE und einer FUNCTION mit CHARACTER-Argumenten in einem C++ Programm sowie die Verwendung einer DLL.

5. `Ex03_C_calls_Fortran` (cpp):

Das C++ Programm kreiert die Struktur

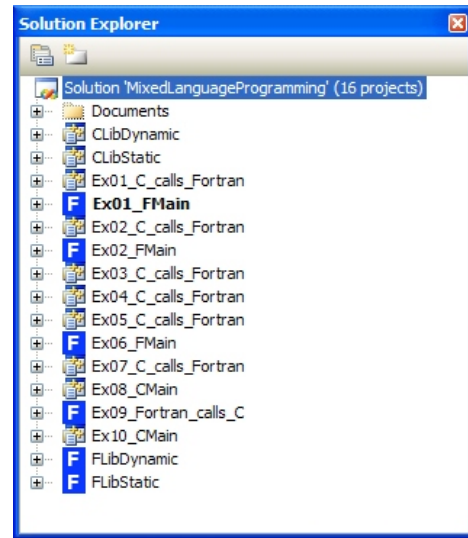


Abb. 7: Die Visual Studio Solution `MixedLanguageProgramming`

```

struct T_PolarCoord
analog zu denen des Fortran MODULE FMixLangMod01
TYPE T_PolarCoord,
führt zusätzlich eine Struktur (struct_doublecomplex) zur
Behandlung von COMPLEXen Zahlen ein und ruft FUNCTIONS des
MODULE
FUNCTION FCxFun02( tPolarCoord ) RESULT(cxRetVal)
FUNCTION FCxFun03( cxVal ) RESULT(tPolarCoord)
aus der FLibStatic.

```

6. Ex04\_C\_calls\_Fortran (cpp):  
Das C++ Programm ruft die  

```

SUBROUTINE FSub04( n, m, iMatrix, rVectorIn, &
                  rVectorOut )

```

aus der FLibDynamic. Es zeigt den Umgang mit Feldern (arrays).
7. Ex05\_C\_calls\_Fortran (cpp):  
Das C++ Programm verwendet globale Variablen, die in dem Fortran MODULE FMixLangMod02 deklariert und initialisiert sind. Es kreiert zudem Strukturen analog denen des MODULEs und zeigt den Zugriff auf deren Komponenten. Das MODULE wird über die statische Library FLibStatic eingebunden.
8. Ex06\_FMain (f90):  
Das Programm greift auf initialisierte Variablen des COMMON Block  

```

COMMON / Ex06CmnBl / cType, rVector, lOnOff, &
                  cName, iMatrix

```

zu, der sich in der statischen Library FLibStatic befindet.
9. Ex07\_C\_calls\_Fortran (cpp):  
Das C++ Programm greift wie das Projekt zuvor auf initialisierte Variablen des COMMON Block  

```

COMMON / Ex06CmnBl / cType, rVector, lOnOff, &
                  cName, iMatrix

```

zu, der in der statischen Library FLibStatic abgelegt ist. Der COMMON Block enthält auch CHARACTER Komponenten. Für den COMMON Block wird in C++ eine äquivalente Struktur erzeugt (struct\_CmnBl), und damit wird der COMMON Block "importiert". Seine Werte werden mithilfe der C++ Funktion CDispCommonVars direkt angesprochen und angezeigt. Eine ähnliche Funktion demonstriert wie man den COMMON Block als Argument einer Funktion verwenden kann (CDispCmnBlStruct). Das Programm weist auf mögliche Probleme bei Verwendung in einer DLL hin.
10. Ex08\_CMain (cpp):  
Das C++ Programm ruft die C++ Funktionen  

```

extern "C" void CSub01( int iArg, float fArg,
                      double &dArg )
extern "C" size_t CFun02( const char* szTextIn,
                          char* szTextOut,
                          const size_t DimTextOut,
                          bool &lInToOut )
extern "C" int* CreateIntArray( int iDim )

```

der statischen Bibliothek CLibStatic, die auch vom folgenden Fortran Programm verwendet werden.
11. Ex09\_Fortran\_calls\_C (f90):  
Das Fortran Programm ruft die C++ Funktionen  

```

extern "C" void CSub01( int iArg, float fArg,
                      double &dArg )

```

```
extern "C" size_t CFun02( const char* szTextIn,
                        char* szTextOut,
                        const size_t DimTextOut,
                        bool &lInToOut )
```

```
extern "C" int* CreateIntArray( int iDim )
```

aus der dynamischen Bibliothek CLibDynamic sowie die Standard C-Funktion

```
size_t strlen( const char *str )
```

der C/C++ Laufzeitbibliotheken. Außerdem wird die Windows Betriebssystem Funktion (WinAPI)

```
void CopyMemory( PVOID Destination,
                const VOID* Source,
                SIZE_T Length )
```

gerufen.

Im MODULE Ex09CInterfaces sind die Schnittstellen (INTERFACES) zu den C++ Funktionen definiert, so daß ein Fortran Programm die C++ Funktionen rufen kann. Gezeigt werden + der Aufbau der INTERFACE Blöcke zum Import der C++ Funktionen unter Verwendung der Direktiven des VF Compilers (!DEC\$ ATTRIBUTES).

+ was bei der Verwendung von CHARACTER Argumenten zu beachten ist

+ wie in VF mit einem C/C++ Pointer zu verfahren ist, der auf eine Speicheradresse zeigt.

Von dem MODULE existiert eine Fortran 2003 basierende Alternative Ex09CInterfacesF2003.f90, die jedoch derzeit nur mit Intel Visual Fortran 10.1 übersetzt werden kann. Um sie zu nutzen ist im Programm Ex09FMain die Zeile 7 auszukommentieren und Zeile 5 zu kommentieren.

## 12. Ex10\_CMain (cpp):

Das C++ Programm ruft die Funktion

```
extern "C" int* CreateIntArray( int iDim );
```

auf, um deren Funktion in C++ zu demonstrieren.

# Index

## !

!DEC\$ ATTRIBUTES	9
!DEC\$ ATTRIBUTES ALIAS	9
!DEC\$ ATTRIBUTES C	9, 11
!DEC\$ ATTRIBUTES REFERENCE	10

## A

Additional Dependencies	7
Adresse	3, 5, 10
ALLOCATABLE	11
Argumentenübergabe	4
Array	
assumed-shape	11
assumed-size	11
deferred-shape	11
explicitely shaped	11
explicite-shape	11
Array Descriptor	11
arrays	14
Assumed-shape Array	11
Assumed-size Array	11
Aufrufkonvention	
C	3, 9
DEFAULT	8, 10
STDCALL	3, 10

## B

base pointer	2
Bibliothek	
dynamische	13
statische	13
BIND	10
by reference	10
by value	10

## C

C	1
C Aufrufkonvention	9
C KINDs	1
C_double	1
C_float	1
C_long	1
C_short	1
call by reference	3
call by value	3
CHAR(0)	11
CHARACTER	
Argument	2, 10
CHARACTER Argumente	15
Character Set	6
CHARACTER-Argumente	13
COMMON Block	14
COMMON Block	14
Compaq Visual Fortran	3
Compiler Direktive	9
COMPLEX	14

CopyMemory	15
CreateIntArray	14, 15
CVF	3

## D

Datentypen	
äquivalente	11
DEFAULT	10
DEFAULT Aufrufkonvention	8
Default Library Search Rules	6
Deferred-shape Array	11
Digital Visual Fortran	3
DLL	13, 14
DVF	3
dynamische Bibliothek	13

## E

EBP	2
EQUIVALENCE	12
ESP	2
Ex01_C_calls_Fortran	13
Ex01_Main	13
Ex02_C_calls_Fortran	13
Ex02_Main	13
Ex03_C_calls_Fortran	13
Ex04_C_calls_Fortran	14
Ex05_C_calls_Fortran	14
Ex06_FMain	14
Ex07_C_calls_Fortran	14
Ex08_CMain	14
Ex09_Fortran_calls_C	14
Ex09CInterfaces	
MODULE	15
Ex09CInterfacesF2003	15
Ex10_CMain	15
Explicit-shape Array	11
Exportdatei	13

## F

Felder	14
Feldtyp	12
Feldvarianten	11
FLibDynamic	13
FLibStatic	13
Fortran 2003	2, 10
FSub01	13
Funktionsprototyp	5, 8

## G

Ganzzahlen	
vorzeichenlos	12
globale Variablen	14

## H

hidden length argument	10
------------------------	----

## I

IFWIN	10
Indiziermechanismus	2

INTEGER	12
Intel Visual Fortran	3
INTERFACE	9, 10, 15
interner Name	5
Interoperabilität	10
ISO_C_BINDING	2
IVF	3
<b>K</b>	
KIND	1
<b>L</b>	
Längenargument	
verstecktes	10, 11
Laufzeitbibliotheken	4, 7
LEN()	11
Library Directories	7
Library-Konsistenz	4
Linker	10
little endian	12
<b>M</b>	
Maschinenprogramm	5
MixedLanguageProgramming.sln	13
Multi-Byte Character Set	6
Multithreaded	4
<b>N</b>	
Namensgebung	4
null terminated string	2, 11
<b>P</b>	
Pointer	15
POINTER	11
Precompiled Headers	7
Project Dependencies	7
Projekte	13
Prototyp	9
<b>R</b>	
REFERENCE	10
Runtime Libraries	4
Runtime Library	7
<b>S</b>	
Schnittstellendefinition	9
Speicheradresse	15
Speichertyp	12
SS	2
Stack	2, 5
Argumentenfolge	3
clean	3
Stack Frame	3
stack pointer	2
stack segment	2
Stackfreigabe	4, 5
Stapelspeicher	2
Startadresse	2, 11

statische Bibliothek	13
STDCALL	3, 10
string	
null terminated	2, 11
zero terminated	2, 11
String	11
strings	2
strlen	15
struct	14
struct_doublecomplex	14
Struktur	13
Symbol	5, 9
Namen	2
<b>T</b>	
T	14
terminierende Null	11
TYPE	14
Typen	
korrespondierende	1
Typkompatibilität	13
Typkonsistenz	4
<b>U</b>	
Unicode	6
unsigned int	12
<b>V</b>	
VALUE	10
Variablen	
globale	14
Variablentypen	1
VB	3
VC	3
VC++ Directories	8
verstecktes Längenargument	10, 11
VF	3
Visual Basic	3
Visual C++	3
vorzeichenlose Ganzzahlen	12
<b>W</b>	
Wrapper-Funktion	12
<b>Z</b>	
Zeichen	
terminierendes	2
Zeichenketten	2
Zeiger	
auf Funktionen	12
zero terminated string	2, 11